Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Sub-Network Access Control Technology Demonstrator: Software Design of the Network Management System

H. Lukasik

The work described in this document was sponsored by the Department of National Defence under Work Unit 1BB22.

Defence R&D Canada - Ottawa
TECHNICAL REPORT
DRDC Ottawa TR 2002-073

Communications Research Centre
CRC-RP-2002-003
August 2002

Canada

Sub-Network Access Control Technology Demonstrator : Software Design of the Network Management System

Lukasik. H.
Communications Research Centre

The work described in this document was sponsored by the Department of National Defence under Work Unit 1BB22

Defence R&D Canada - Ottawa

Technical Report
DRDC Ottawa TR 2002-073

Communications Research Centre

CRC-RP-2002-003

AUGUST 2002

Abstract

This final report provides a record of the results of the network management portion of the Sub-Network Access Control Technology Demonstrator project. It summarises the concepts behind this development effort, and describes the completed design and implementation work. This project is part of an on-going effort to build an IP based network using wireless technologies. However, the goal behind this project remains the production of an exploratory prototype. In other words, it is one more step towards the goal of transitioning wireless technologies to the Canadian Operational Fleet.

The proposed IP network presents several unique challenges to network management, due to its low bandwidth wireless links and continually changing topology, that existing products have not been designed to handle. These challenges are explored in order to provide a better understanding of the requirements they impose on network management. A network management tool design is then proposed and the implementation of its prototype is described.

Résumé

Ce rapport vise à préserver les réalisations des travaux reliés à la gestion de réseaux informatiques du projet de démonstrations de la technologie de contrôle d'accès à des réseaux subordonnés. Il enregistre les concepts reliés à ces travaux et présente la conception et la mise en oeuvre qu'ils ont engendrés. Ce projet fait partie d'efforts soutenus visant à construire avec des technologies radio un réseau informatique basé sur le protocole IP. Cependant, le but de ce projet demeure la création d'un prototype exploratoire. En d'autres mots, ce projet est une étape supplémentaire dans le dessein de migrer ces technologies radio vers la Flotte Opérationnelle Canadienne.

Le réseau informatique base sur le protocole IP en développement possède les caractéristiques d'avoir des liens radio à bande passante restreinte ainsi qu'une topologie qui change constamment. Ces caractéristiques propres à ce réseau constituent des obstacles qui ne permettent pas d'utiliser les produits de gestion de réseau existants puisque ces derniers n'ont pas été conçus pour y répondre. Ce document commence par scruter ces obstacles afin d'en déduire les besoins spécifiques qu'ils exigent d'un système de gestion de réseaux informatiques. La conception d'un outil de gestion de réseaux informatiques est ensuite proposée et la mise en oeuvre d'un prototype décrite.

This page intentionally left blank

Executive Summary

This final report provides a record of the results of the network management portion of the Sub-Network Access Control Technology Demonstrator project. It summarises the concepts behind this development effort, describes the completed design and implementation work, and concludes by speculating on future research areas. This project is part of an on-going effort to build an IP based network using wireless technologies.

Military Significance

The overall goal for the proposed network is to provide improved information exchange between military units by promoting Command, Control, Communications, and Computers (C4) inter-operability through the extension of IP based networking both at sea and in the littoral regions. This network will have a continually changing topology as it is divided into several sub-networks each of which are frequently added or removed from the network. Each of these sub-networks belongs to its own military unit, and is added or removed from the network as the unit deploys or scales down its actions during the different phases of an operation. Furthermore, the network is being built with low bandwidth wireless links, which can easily be overwhelmed by peaks in messaging requests.

The characteristics of the proposed network present several unique challenges to network management that existing products have not been designed to handle, as they assume that the network being managed consists of high bandwidth links and has a slowly evolving topology. Existing network management products often use continual polling to detect new or no longer active devices, or they make many Simple Network Management Protocol (SNMP) requests to gather performance statistics. This is especially true during the initialisation of the network management stations. These mechanisms cannot be used in the proposed network as they would consume too much of the low bandwidth links' capacity. The general guideline used in this design, is that management traffic, just as any other type of traffic, must be minimised as much as possible on low bandwidth links. Generic solutions, such as using multicast addresses, cannot be used, as they do not prevent a network manager from overwhelming a link with requests. The particular nature of the proposed network has necessitated the development of a special network management solution.

The basis for this project was to make a preliminary attempt to provide network management tools that would allow the deployment of Sub-Network Access Controllers (SNAC)[i] card technology in the field while keeping in mind the limitations of the proposed network. The tools available at the start of the project for controlling and monitoring SNAC cards didn't allow the user to easily obtain a view of the cards' status and performance, as they required detailed knowledge of the technology beyond what can be expected of operators in the field. Nevertheless, the goal for this project remained the production of an exploratory prototype; as such a prototype would also provide a platform on which to test potential solutions to the issues that limit the use of commercially available network management systems in the target network. In other words, this project is one more step towards the goal of transitioning wireless technologies to the Canadian Operational Fleet.

Requirements

The proposed network management solution must provide the normal monitoring and configuration mechanisms generally found in existing products. However, experimental devices such as the SNAC cards must also be supported, along with specific mechanisms to enforce military policies, such as

---

[i] The SNAC is a specialised card that manages access to lower bandwidth communication links. It implements the layer two of the wireless link it supports and it is thus responsible for managing the link's communication such as managing transmission collisions, retransmissions, or adding/removing remote devices etc.

the ability to take offline only the outgoing side of specific radio links. The goal of such a mechanism is to allow the reception of information downloads while avoiding detection by enemy forces due to a unit's radio emissions. Furthermore, each deployed unit must be able to function independently, and in so doing must manage their own network. Each unit thus requires it's own network management console. Any solution must therefore also provide the ability to remotely assist the troubleshooting of deployed units' local networks.

Results

In order to fulfil these requirements a design for a special network management tool is proposed, and the initial implementation of its prototype is outlined. The design places elements in three distinct locations in the network, each element having its own responsibilities. The three locations are: the network management Agents on the devices themselves, the Local Management Console on the Local Area Network (LAN) of the managed devices, and the Remote Management Application (which can reside anywhere in the network). In each of these locations tasks are allocated to different design elements. The network management Agents have two design elements: the Agent itself, and the SNAC controller that implements the existing control functions of the card upon which the Agent has been added. The Agent also has a data repository known as the Management Information Base (MIB). On the Local Management Console the responsibilities are divided between a user process known as the Management Console Graphical User Interface (GUI), and three daemon processes known as the Log Manager, the SNMP Session Manager, and the SNMP Trap Monitor. The Local Management Console also has two main data repositories: the operational data logs, and the historic data logs. Since no prototyping work was performed on the Remote Management Application it remains undivided and is therefore currently a single design element.

Although not every element of the proposed design has been fully implemented, several sections were successfully demonstrated, notably SNMP Agents for the experimental cards, and an initial version of the Local Management Console. This document outlines the implementation of the elements that have been prototyped.

Recommendations for Future Work

This document concludes by proposing several avenues for future development, and outlines some potential solutions that should be prototyped to evaluate their value. The avenues proposed target three specific area of interest. Firstly, developing an automatic device discovery algorithm that uses inherent knowledge of the network being monitored to perform its work without overloading low bandwidth links. Secondly, reducing the amount of traffic on low bandwidth links caused by remote network management. Thirdly, providing a basic network management solution that could be used without the integration of commercially available network management products.

Sommaire

Ce rapport vise à préserver les réalisations des travaux reliés à la gestion de réseaux informatiques du projet de démonstration de la technologie de contrôle d'accès à des réseaux subordonnés. Il enregistre les concepts reliés à ces travaux et présente la conception et la mise en oeuvre qu'il a engendrée. Ce projet fait partie d'efforts soutenus visant à construire avec des technologies radio un réseau informatique basé sur le protocole IP.

Importance stratégique militaire

Le but visé du réseau informatique en développement est l'enrichissement de l'information échangée dans les communications entre les unités navales et côtières grâce à leur inclusion dans un réseau basé sur le protocole IP. Ce réseau est subdivisé en plusieurs réseaux subordonnés qui individuellement correspondent à une unité militaire. Cette division apporte au réseau la caractéristique d'avoir une topologie qui change constamment puisque les réseaux subordonnés s'y ajoutent ou s'en retirent quand l'unité à lequel ils correspondent est déployée ou démobilisée pendant les différentes phases d'une opération militaire. De plus, le réseau est construit de liens radio avec une bande passante restreinte dont la capacité peut aisément être dépasse par une succession rapide et soudaine de requêtes.

Les caractéristiques propres au réseau proposé constituent des obstacles qui ne permettent pas d'utiliser les produits de gestion de réseau informatique existant puisque ces derniers sont bâtis dans l'optique qu'ils opèrent sur des liens avec une large bande passantes dont la topologie évolue lentement. Les produites commerciaux de gestion de réseaux informatiques utilisent des mécanismes qui sondent continuellement sur le réseau pour y découvrir de nouveaux appareils, détecter les pannes d'appareils ou ramasser des statistiques sur le comportement du réseau informatique. Ces mécanismes possèdent le défaut de consommer beaucoup de bande passante sans tenir compte de la capacité des divers liens sur lesquels transitent les messages que ceux-ci gênèrent. De plus, cette situation est aggravée par le supplément de ces messages qui est généré lorsque la station de gestion de réseau informatique est démarrée. Bref, le fils conducteur de ces problèmes est que la quantité de messages de gestion de réseau informatique, ainsi que de n'importe quel autre type, doit être minimisé autant que possible sur les liens à faible bande passante. Des solutions génériques comme l'utilisation d'adresses à multiples destinataires peuvent être utilisées, mais celles-ci n'empêchent pas une station de gestion du réseau informatique d'accaparer toutes la bande passante d'un lien avec ses requêtes. Il est donc devenu nécessaire de développer une solution aux problèmes reliés à la gestion d'un tel réseau informatique.

Le but de ce projet est donc de produire la version initiale d'un ensemble d'outils de gestion de réseaux informatique qui ultimement permettront de livrer les cartes de Contrôle d'accès aux Réseaux Subordonnés[ii] (SNAC) aux usagers, et ce sans perdre de vue les caractéristiques propres du réseau visé. Les outils disponibles au début du projet pour le contrôle et la supervision de la performance de ces cartes ne permettaient pas d'en obtenir aisément une vue d'ensemble. Ceux-ci exigeaient une expertise de la technologie en question et la compréhension nécessaire de ses concepts dépassait largement les exigences usuelles auxquelles les opérateurs sont normalement soumis. Néanmoins, le produit résultant de ce projet demeure la création d'un prototype qui fournit une base sur laquelle des solutions aux problèmes reliés à l'utilisation des systèmes de gestion de réseaux informatiques sur le réseau visé pourront être validées. En d'autres mots, ce projet est une étape supplémentaire dans le dessein de migrer ces technologies radio vers la Flotte Opérationnelle Canadienne.

[ii] Les cartes de Contrôle d'accès aux Réseaux Subordonnés (SNAC) sont des cartes spécialisées qui contrôle l'accès aux liens de communication à bande passante restreinte. Elles implémentent la deuxième couche du protocole du lien radio qu'elles supportent; elles ont donc la responsabilité de gérer les collisions et les retransmissions des messages ainsi que l'intégration ou le retrait de sites qui communiquent à l'aide de ce lien.

Besoins spécifiques

La solution de gestion de réseaux informatiques envisagée doit fournir les outils habituels de configuration et de surveillance de réseaux informatiques. Cependant, les appareils expérimentaux que sont les cartes de Contrôle d'accès aux Réseaux Subordonnés (SNAC) ainsi que des mécanismes spécifiques permettant de désactiver les émissions radio d'un lien radio sans pour autant arrêter la réception de messages par ce même lien doivent êtres supportés. Le but de ces mécanismes spécifiques est de permettre le télé-déchargement de données tout en évitant la détection de l'unité par des forces ennemies grâce à ses émissions radio. Finalement, chaque unité militaire doit être capable de gérer de façon indépendante son propre réseau informatique; chaque unité doit donc posséder sa propre station de gestion de réseau informatique. Par contre, il doit également exister des mécanismes pour permettre d'assister à distance la résolution de problèmes reliés aux réseaux informatiques des unités déployées.

Réalisations

Pour répondre à ces besoins spécifiques, la conception d'un outil de gestion de réseaux informatiques est proposée et sa mise en oeuvre entreprise dans un prototype. Les diverses composantes avec leurs responsabilités propres sont reparties dans trois endroits différents; Ces endroits sont: sur les appareils étant gérés, sur la console locale de gestion de réseau informatique qui est sur le réseau local (LAN) des appareils étant gérés et une application de gestion de réseau à distance qui peut être placé n'importe ou sur le réseau. Dans chacun de ces endroits, les responsabilités sont distribuées à différentes composantes. Pour la console locale, les responsabilités sont distribuées parmi le processus du GUI et les processus serveurs du gestionnaire des records historiques, le gestionnaire des sessions SNMP et le surveillant des TRAP SNMP. La console locale possède également deux dépôts de données qui contiennent respectivement les données opérationnelles et les données historiques. Les appareils étant gérés possèdent également deux composantes, soit le contrôleur lui-même qui réalise les fonctions de l'appareil et l'agent SNMP. L'agent SNMP possède également un dépôt de données, soit le MIB. Finalement, l'application de gestion à distance demeure en une seule composante puisque aucun prototype n'en a été réalisé.

Bien que ce ne sont pas toutes les composantes de l'outil de gestion de réseaux informatiques qui sont implémentées, plusieurs fonctionnalités ont été démontrées avec succès; en particulier, les agents SNMP pour les cartes expérimentales ainsi qu'une version préliminaire de la console locale de gestion de réseau informatiques. Ce document présente l'implémentation des composantes qui ont été démontrées avec un prototype.

Recommandation de Travaux Subséquents

Ce document conclu en proposant plusieurs directions que devraient prendre les efforts de développement ultérieurs ainsi que plusieurs solutions potentielles dont la valeur devrait être étudié grâce à de l'expérimentation par prototypage. Ces directions visent trois domaines d'intérêt spécifique. Le premier domaine est relié à l'invention d'un algorithme de découverte automatisé des appareils existant sur le réseau informatique qui utiliserait des connaissances inhérentes à la composition du réseau et qui ne surchargerait pas les liens avec une bande passante restreinte. Le second domaine est relié à l'utilisation de techniques visant à réduire la consommation de la bande passante par la gestion du réseau informatique à partir d'un poste de travail situé de l'autre côté d'un lien avec une bande passante restreinte. Et finalement, le troisième domaine est relié aux éléments de développement qui resteraient à réaliser afin de fournir une solution de base de gestion de réseau informatique qui fournirait assez de fonctionnalité pour être utilisé sans l'apport de produits commerciaux.

Table of Contents

List of Figures

List of Tables

1- Introduction

The design presented in this document is part of an on-going effort to build an experimental IP based network using emerging wireless technologies. This network is being built to further refine and showcase the technologies used in the annual Joint Warrior Inter-operability Demonstrations (JWID)[iii] and the Communication System Network Inter-Operability (CSNI) Navy Network Trials. In short, development of the Technology Demonstrator is one more step towards the goal of transitioning these technologies to the Canadian Operational Fleet.

The network management portion of this effort is of primary concern since, during experimentation in previous JWID, it was determined that existing network management products such as HP Open View™ were inadequate when operating over low bandwidth links. The primary reason for their failure being that these tools were built to manage networks comprised of high bandwidth links, and therefore they were generating too much management traffic for use with low bandwidth wireless links. The particular nature of the target network required that a network management solution adapted to it be developed.

The goal of this document is to summarise the network management portion of the Sub-Network Access Control Technology Demonstrator (TD) project, to detail the concepts behind this technology, and to present the design and implementation work currently completed. The goal behind this project remains the production of an exploratory prototype. The implementation described in this document provides a starting point on which to base future development.

This document is structured as follows: Section 2 presents the target network and the special requirements it places on network management solutions. Section 3 presents the high level design. Section 4 presents the design and implementation of the Sub-Network Access Controller[iv] (SNAC) card Network Management Agents. Section 5 presents the design and implementation of the Network Management Console. Section 6 presents some thoughts on the direction of future study and development. Finally, section 7 concludes with an analysis of the project's achievements.

[iii] JWID is an inter-operability demonstration of new communication technologies supporting different units in a multinational task Group. It is an annual opportunity for the military to be exposed to efforts under development, and for industry and R&D activities to receive military comment early in any development effort.

[iv] The SNAC is a specialised card that manages access to lower bandwidth communication links. It implements the layer two of the wireless link it supports and it is thus responsible for managing the link's communication such as managing transmission collisions, retransmissions, or adding/removing remote devices etc.

2- Requirements

This section discusses the requirements imposed by the target network on any network management solution. Specific issues are explored and the constraints set for the design presented in this document are outlined. The following subsections describe the intended use of the network, the network topology, the requirements specific to the target network, and the network management strategy that will be used for the proposed design.

2.1- Description of the Target Network

The overall goal for the network is improved Command, Control, Communications, and Computers (C4) effectiveness, and enhanced maritime information exchange with joint and combined forces, including amphibious operations, by promoting inter-operability through the extension of IP based networking both at sea and in the littoral regions forming a Task Group Area Network (TGAN). The TGAN is a subdivision of the overall network, the Coalition Wide Area Network (CWAN), which consists of all the units involved in an operation. The TGAN is further divided into several sub-networks, each of which represent Autonomous Systems (AS) that are dynamically added or removed from the network as the corresponding units deploy or scale down their actions during the different phases of an operation. An AS has the capability to add, remove, or modify its links to the TGAN, or even directly with the CWAN, as needed during the different phases of an operation. Each AS is further divided into Local Area Networks (LANs). The TGAN is connected to the CWAN through specific gateway nodes.

The IP based network being tested must have several key capabilities, which include:

- The support of multicast mobile IP networking over multiple sub-networks with an emphasis on space based connectivity;
- The support of multicast messaging through the mobile IP network linking military units;
- Access to distributed secure databases on an "as required" basis;
- The ability to include low bandwidth radio links, such as Ultra High Frequency Satellite Communications (UHF SATCOM), High Frequency Beyond Line Of Sight (HF-BLOS) and High Frequency Extended Line Of Sight (HF-ELOS);
- The ability to allow easy and quick reconfigurations based on input from an AS manager.
- The support of Emission Control (EMCON) mechanisms.

EMCON is a mechanism through which radio silence can be enforced by some units while they remain connected to the network and receive data downloads. In other words, a unit in EMCON mode would stop sending transmissions but would still listen for broadcast messages or messages directly sent to it. Radio silence may be necessary in some instances to avoid detection by enemy forces. However, radio silence at a node can be total or partial depending on the nature or level of the perceived threat. For example, a node in partial radio silence could stop sending HF radio transmissions that propagate in a wide area, but maintain a narrow directional up-link with a satellite.

2.2- Network Topology

An example topology for the target network is shown in Figure 1. It is based on the TGAN architecture used during the JWID exercises. The network consists of a collection of links between various naval units, a satellite, and shore bases. The types of links in use include UHF SATCOM, HF-BLOS, and HF-ELOS.

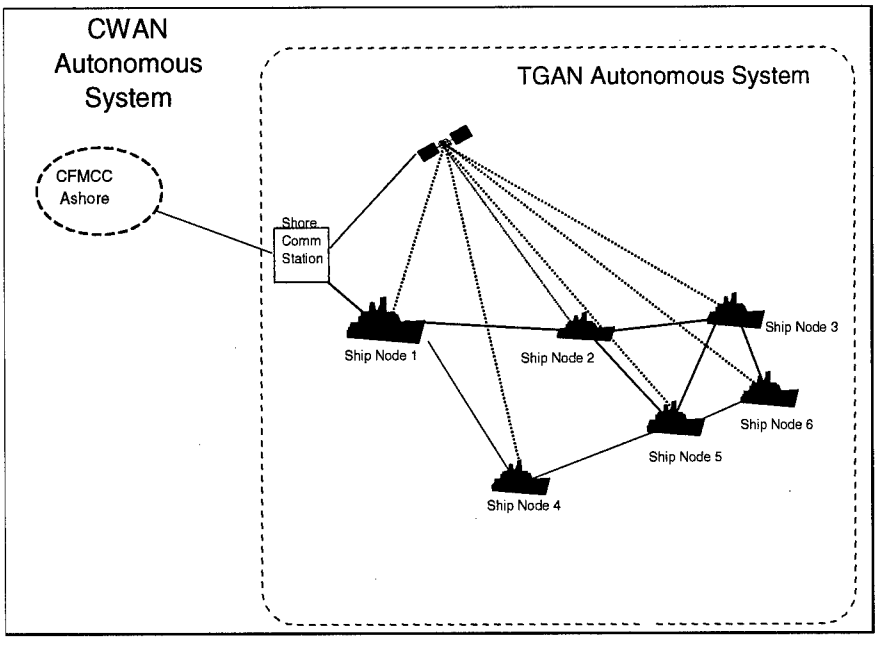


Figure 1 *Example Network Topology*

Each of the maritime units in the network possesses an Ethernet LAN connected to radio links as shown in Figure 2.

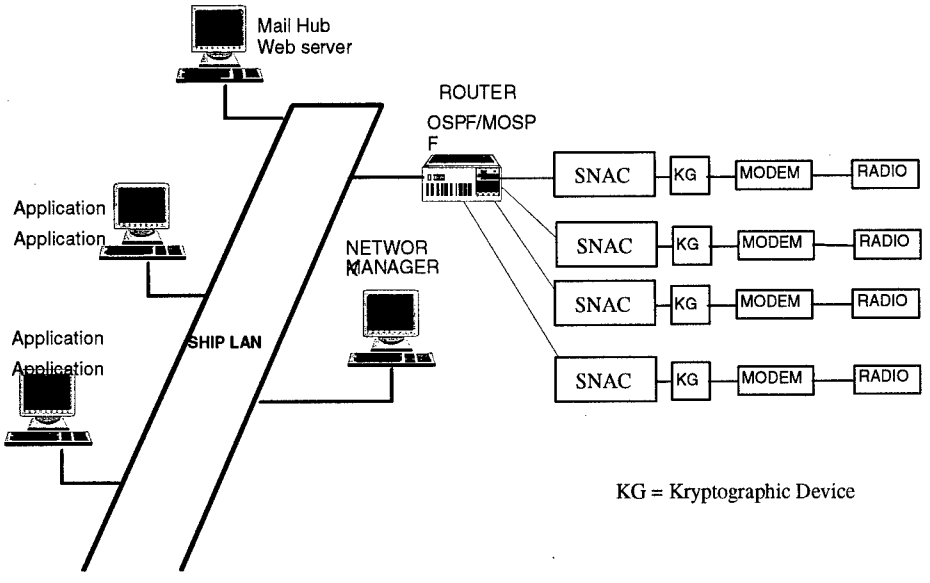


Figure 2 *Generic ship node architecture*

2.3- Management Requirements

This subsection presents the requirements and the design objectives of the network management system. Each of the five network management Functional Areas (FCAPS[v]) is explored in turn. Broadly speaking, the network management tools need to support a user-configurable Graphical User Interface that allows the user to remotely manage all of the network devices within the same AS from a single terminal. The proposed network management solution provides the following functions:

- Display the local network and the TGAN topology along with statistics compiled from devices' Management Information Bases (MIB);
- Monitor network connectivity and current network performance;
- Display the status of the network components;
- Configure, monitor, and control (including the ability to reboot) routers, other SNMP-managed network devices, and SNAC cards;
- Support Emission Control (EMCON) on either a link-by-link, node, AS, or network basis;
- Remotely assist deployed units to troubleshoot their networks with IP based tools that can remotely control components of the problem network;
- Automatically generate network status reports, at user-selectable intervals, which will be accessible to all TGAN users using a graphical application.

2.3.1- Fault/Problem Resolution Requirements

This subsection presents the requirements of the target network in the fault/problem resolution functional area. The fault/problem management functional area normally includes two main functions: fault isolation and diagnosis, and restoration of the system. In short, an operator will be alerted to failures or degrading conditions, and tools will be available to allow him or her to make the changes required to keep the network working at its peak performance.

Network management solutions to fault isolation and diagnosis typically divide the functionality into three components: the device monitoring probes or the MIB variables, the monitoring agents, and the network management console. In this division of tasks, software agents observe the MIB variables and send SNMP events (known as Traps in SNMP-V1) to the Management Console when specified thresholds are crossed. The monitoring agent may be on the device itself, at the network management console, or at some middle point in the network. These different options for the location of the monitoring Agent allows the network designer to strike a balance between the complexity and cost of the devices and the bandwidth consumed by monitoring them. Furthermore, a network may be managed by either a central management console or by many consoles, each using different responsibility distribution schemes (for example peer-to-peer, or hierarchic).

There are two key requirements in fault isolation and diagnosis resolution specific to the target network that are important to the selection of the network management architecture. The first requirement is that each AS must be able to function independently. As such, the network management responsibilities follow AS divisions, and each AS requires its own network management console. The second requirement is the ability to remotely assist deployed units (or AS) troubleshoot their networks.

Given the two requirements listed above, it would seem to be convenient to adopt the peer-to-peer network management architecture. However, one must consider the fact that AS may be connected to the rest of the network through low-bandwidth radio links or expensive satellite links. This necessitates the reduction of unnecessary network management overhead and bandwidth use on gateway links to a greater

---

[v] The acronym FCAPS is used to represent the key elements of the ISO network management model: Fault management, Configuration management, Accounting management, Performance management, and Security management.

extent than for typical networks. For this reason, information such as events will need to be resolved locally at the AS's network management console or filtered and correlated with other events or collected statistics as much as possible before being sent outside an AS. An independent management and monitoring console for each leaf AS (or deployed unit) is therefore the chosen approach for the fault isolation and diagnosis functionality. A network wide management console will be responsible for any resolution and correlation of events with other events or collected statistics that an AS could not resolve itself.

Restoration of a system generally requires many tools to provide enough information to allow proper diagnosis of problems, as well as tools used to restore the network to a normal state. Similarly to the diagnosis functions, the key requirement here is the ability to remotely assist deployed units (or AS) troubleshoot their networks. It was decided that this requirement would be fulfilled with the use of a graphical tool that will access data only when required.

From this discussion and the requirements caused by the network topology one may observe that the elements used to implement the required behaviour must reside in three distinct locations in the network:

1. On a remote machine in the CWAN resides the Remote Management Application that contains information retrieval tools as well as network performance monitoring and troubleshooting analysis tools. The application will be able to:
   - Retrieve and set MIB values;
   - Remotely offline, reset, and reboot devices;
   - Provide different remote analysis tools, such as tools to test the end-to-end connectivity of specific paths or to capture packets for off-line protocol analysis.
2. On the managed devices' LAN resides a Local Management Console that will be responsible for the management of the devices in an AS. This console will provide the following functions:
   - All of the functions of the Remote Management Application;
   - Set thresholds for events;
   - Poll local devices and generate ensuing events (could be delegated to the agents of other devices if they have these capabilities);
   - Receive, filter, and correlate events;
   - Forward the events that could not be resolved locally;
   - Display a simple picture presenting the status of the local devices as well as those of the TGAN.
3. On the physical devices (SNAC cards, routers, etc.) themselves resides an agent that will provide the following functions:
   - Store and update the MIB values;
   - Generate SNMP Traps;
   - Execute commands received via SNMP messages;
   - Periodically send status information (if the existing agent implements the capability).

2.3.2- Performance Management Requirements

This subsection presents requirements that belong to the performance management functional area. This functional area refers to the gathering of historic data for off-line analysis in order to allow better growth planning or for analysis of the current configuration's performance over time. There are two main components in this functional area. The first component accesses the MIB values or captures the data as it flows in the network and stores it for later retrieval. The second component retrieves the data and provides different analysis and display functions.

The requirements specified in this functional area are not specific to the target network with the exception of the need to generate reports on the current network's performance at user selectable intervals.

These reports must later be accessible via a graphical application. The only major issue that must be considered in implementing a solution to these requirements is the same as that for the fault and problem resolution functional area. That is, to relieve the low bandwidth links of unnecessary network management bandwidth overhead . As it is not practical to broadcast all of the collected status and performance data outside a leaf AS, it must be stored locally and made available to the rest of the network only when required. Automatically generated reports will also be generated and stored locally at the AS's management console and may be accessed via a graphical interface.

The functionality in the performance management functional area will be divided as follows:

The Local Management Console will provide the following functions:

- Periodically poll local devices for specific MIB values and archive them;
- Capture events and status information as it flows through the network for which it is responsible;
- Periodically generate and store network status reports at user selectable intervals.

Both the Local Management Console and the Remote Management Application will provide the following functions:

- Retrieve and display historic MIB values, events, and other captured information;
- Retrieve and display the network status reports.

The agents will provide the following functions

- Calculate MIB performance values, and other necessary information, and send them either periodically or upon request;
- Generate events based on predefined criteria.

2.3.3- Configuration Management Requirements

This subsection presents the requirements in the configuration management functional area that are specific to the targeted network. Configuration management usually consists of maintaining a device inventory database and setting parameters on devices.

The requirements in this functional area are fairly standard when strictly following the definition above, and can simply be placed within the management console's purview. However, there are specialised requirements in the functional area of configuration management that are derived from the network's architecture and use. There are two main needs that must be fulfilled with specific functionality, both related to managing changes in the network topology. However, they differ in their fundamental focus. Each of these is presented in the following subsections.

2.3.3.1- Autonomous systems as a single unit

The first set of configuration management requirements are related to the fact that the network is divided into many AS, and each AS can itself contain smaller AS thus forming a tree hierarchy. An AS can be added or removed from the network or change the link by which it is connected to the network. These changes will take place at differing moments in time as the corresponding units deploy, establish their positions, or scale down their actions during the different phases of an operation. These changes could also occur when a unit moves in or out of the radio transmission range of other units. Another point to consider is that from the point of view of the routers all of the devices in an AS will be added, removed, or their links changed in the network as a single entity. Such topology changes are required more frequently than in an average network, and it is for this reason that specific tools are required to ease these changes.

Typical auto-discovery mechanisms, such as the constant polling for new devices, cannot be used in the target network since they consume too much capacity on the low bandwidth links interconnecting AS. In addition, due to the nature of radio transmission characteristics, these links can be relatively unstable as they reach radio transmission range limits, or when they are affected by other phenomenon such as weather. These two facts make it unsuitable to transmit the existence and characteristics of all of the AS's

devices each time an AS is added to the network, or to have the devices of an AS ping many of the devices in another AS when the link between them goes down.

A solution to this issue could be to create a table that contains a description of all the devices in each AS that takes part in an operation. This table would be used to add or remove devices, or to change the links used to reach the devices of an AS, in a single operation. It is important to note here that a specific means to add to this table would have to be devised.

Another way to save bandwidth would be to use scripting capabilities. Such capabilities could provide several benefits since topology changes will tend to be planned before the start of an operation. These benefits include a reduced chance of error in entering the commands, the ability to test the validity of a topology alteration before its actual use in combat, and the ability to simply provide an automation of command sequences to produce the desired result. These scripts could be invoked by an operator or at a previously scheduled time.

Finally, the use of GPS equipment could be a solution to solving the issue of detecting whether a unit is within radio transmission range of other units so that the link used between them may be changed. One must note that it is uncertain if such a solution is feasible or if any gains can be expected over the simple solution of testing the radio link at regular intervals.

In summary, the Local Management Console should be able to:

- Use a table that contains the description of the devices and the routing information for an AS to reduce network management traffic;
- Use scripts to modify the network's topology;
- Automatically invoke scripts at a scheduled time;
- Use GPS information to automatically change links used between two or more AS.

2.3.3.2- Emission Control

The focus of the second set of configuration management requirements is to support emission control mechanisms. To support these mechanisms the network management system must, at a minimum, be able to offline only the outgoing port of specific links while keeping their incoming port active as specified by the EMCON policy. Once this minimum requirement is satisfied, the invocation of different EMCON levels can be supported by several means. These go from the most basic, such as having the network manager manually apply the changes to every link, to the more sophisticated, such as allowing the operator to change the EMCON setting from the console, causing actions to be executed on all applicable links. The automated actions could be defined to act on either specific links or on all the links of a specific type at a node, a sub-network, or an AS. Furthermore, the type of links that need to be put off-line for a given level of EMCON could be predefined. A mechanism to remotely access the EMCON functions would also be a desirable feature, as it would allow an AS to set the EMCON level of all of its children AS. These mechanisms should preferably be available from a graphical interface.

In summary, the emission control mechanisms functionality will be divided as follows:

The Local Management Console will be able to:

- Include the EMCON level status of each link in the network's status display as well as in network status reports.

Both the Local Management Console and the Remote Management Application will be able to:

- Offline only the outgoing port of a link while the incoming port remains active;
- Select an alternate link for outgoing messages when the outgoing port of the primary link is off-lined;
- Automate actions to set the state of links based on simple EMCON level settings that may be changed through a graphical interface. Either a specific link, or all of the links of a specific type at a node, a sub-network, or an entire AS may be affected;

2.3.4- Security Management Requirements

This subsection presents the requirements in the security management functional area that are specific to the target network. Security management usually consists of password administration, authentication, data encryption, and security audit log. As in any network, the solution requires network security functions at many levels.

The first area of concern is access control via login protection and user profiles. Both the graphical interface and the management console require a login with a password. That having being said, the mechanisms to provide access to the network are no different from any other system, and it is therefore of no interest to put any more emphasis on the subject.

The second area of concern is data encryption. Using cryptographic units on all links would fulfil this requirement. The only source of concern is that it must be possible to manage these units through the network management interface. There is also a need to define what type of management is required, if any.

The third area of concern is the ability to log all actions in order to ensure that any security breach can be traced and properly blocked in the future. To this end, each management console will log all login attempts and all actions that are performed from it. Furthermore, the agent responsible for serving remote requests on an AS will log every action it performs along with the User Id of the requestor.

A final possible security feature would require that a node be in the auto discovery table described earlier before it can be added to the network. The table entry could also be extended to include the public key of the node in a public/private key mechanism to allow further encryption and signature of data transmitted on the network. The public key could also be used to encrypt management functions.

The following functionality is being considered for both the Local Management Console and the Remote Management Application:

- Access control via log-in protection and user profiles;
- Mechanisms to modify user profiles and passwords;
- Management of cryptographic units;
- Logging of every login attempt and of all actions taken by a management console;
- Logging of every action taken on behalf of a remote user by each agent that serves remote requests.
- Public/private key to be included in the auto discovery table and used to control access to management functions.

2.3.5- Accounting Management requirements

Accounting management usually consists of gathering usage statistics for billing purposes. This subsection would normally present the requirements in the accounting management functional area that are specific to the targeted network. However, there are no such requirements presently envisioned in the target network.

2.4- Network Management Strategy

This subsection expands on the requirements described in the previous subsection by exploring in more detail the underlying components of the network, as well as the status information and functions that are required from them. The existing management elements as well as the specific constraints that were used to reach the proposed design are also discussed in order to demonstrate how the requirements for the network management solution were translated into actual development items.

The first subsection describes the devices that need to be managed. The second subsection presents the constraints that were placed on the design to limit the cost of development and ownership of

the proposed design. Finally, the third subsection discusses the information (i.e. MIB variables) required from the devices by the management policies.

2.4.1- Managed devices

The network uses several different types of wireless links to connect remote AS. For simplicity, only leaf AS are considered in this discussion. Each of these AS will be composed of one or more LANs where one workstation is responsible for local management.

Each link is accessed and controlled by the routers through its own SNAC card (as shown in Figure 2). The SNAC cards are responsible for managing communications on their respective links, and perform tasks such as reacting to transmission collisions, performing retransmissions, and adding and removing remote devices; they are also responsible for implementing the layer two protocol of the wireless link. The SNAC cards are mounted inside VME card cages that provide them with shared memory, disk access, a communication bus, and regulated power. Each of the SNAC cards contain a Sub-Router Interface Unit (SRIU) that is responsible for the interface's management functions and is used for managing, monitoring, and collecting data from the controller functions.

The different wireless links that must be supported are UHF SATCOM, ELOS, and BLOS. There are also ISDN links where wire based links are possible. All of these links are supported by a small number of different SNAC cards.

Each LAN is fairly standard in the sense that it consists of PCs running Microsoft Windows™ and Unix workstations. There is also a timeserver whose responsibility is to provide correct time synchronisation for the LAN. In particular, it allows the SNAC cards on each end of a link to synchronise the start of periodic link status checks. All of these components are connected together through an Ethernet hub and a router that routes messages to the appropriate external links. The normal path for messages is from the workstation to the router, through the hub, and then to the SNAC cards managing the target external link. The same route is used to perform local network management functions as well as to load the operating system onto the SNAC cards

In summary, the Network Management Application must be able to manage all of the equipment that constitutes the building blocks of the network. The Network Management Application must support the following devices:

- SNAC cards,
- Routers to the external links,
- Cryptographic units for each link,
- Modems or radio units depending on the type of link,
- GPS receiver NTP LAN time servers,
- Hubs,
- Host computers (PCs, UNIX Workstations (HP, SUN, etc.)).

It is important to note that the SNAC cards are prototype devices with very basic network management functions and that these must be enhanced. Routers are also very important in a network where specific requirements, such as support for EMCON, must be considered. The other devices listed above have either limited interest to our study, or the available device agents provide all the functionality we require. It is for these reasons that the SNAC cards and the routers will be the focus of the design and implementation study presented in the remainder of this document.

2.4.2- Design Constraints

This subsection presents the general strategy the Local Management Console and the Remote Management Application will use to manage the different devices. In short, it discusses the various

constraints that were imposed on the design and that could not be associated with a single network management functional area.

One of the main issues this project must consider is the requirement to limit as much as possible the traffic generated by the network management functions on the low bandwidth links. To deal with this issue, the responsibilities of the fault and problem management (polling, event filtering, and alarm management), performance management (periodic polling for performance data and safeguarding it in a data repository), and security (password management, authentication, and keeping logs) have almost all been assigned exclusively to the Local Management Console.

To further limit the bandwidth overhead while still allowing extensive post-mortem analysis, it was decided that two types of logs were needed: historical data logs, and operational data logs. The purpose of the operational data log is to allow the network operators to evaluate the current performance and see which events have brought about a given state. Only a filtered version of the recent events will be kept to reduce the log size in order that a smaller amount of information must be transferred when the log is requested by the remote management facility. The purpose of the historic data log is to allow the analysis of all of the events that have occurred in the course of an operation. This log will therefore include not only the same data as the operational log, but could also hold other data necessary to reproduce the conditions encountered by the unit (for example reports on weather conditions limiting the transmission capacity of the wireless links). Furthermore, the logs would keep all of this historic data until a unit could download that data over a high bandwidth wire link. The exact content of these logs has not yet been determined. However, it is understood that the logs will include performance metrics as well as a complete log of all actions taken by the operator at the Local Management Console.

There was a question as to whether it was necessary that the Management Console collect performance statistics and capture events (i.e. Traps) at all times or only when the network manager was logged in. After due consideration, it was deemed necessary that statistics should be collected at all times.

The Remote Management Application will access the operational data on a need to know basis using the File Transfer Protocol (FTP). For security reasons, as well as for logging purposes (since all logging will take place exclusively at the Local Management Console), the Remote Management Application will only be able to access an agent's data through the Local Management Console. The Local Management Console will be the only component able to talk with the agents and will use the SNMP protocol to do so. The Remote Management Application will use FTP to access the operational logs, and the User Data Protocol (UDP) to contact the Local Management Console.

Finally, another design goal is to minimise the amount of work required to support each device type. Furthermore, the solutions selected must reduce ownership costs as well as facilitate development. Because of these goals, several design decisions were made. The constraints that have been imposed on the design to reach these goals are listed below.

- The standard protocol SNMP will be used to implement the communication between the Local Management Console and the Agents. However, for greater security and compatibility with upcoming standards, SNMP-V3 will be used [2];
- Standard MIB-II [3] definitions will be used as much as possible, and adding MIB definitions in the enterprise sub-tree will be used only as a last recourse.
- To reduce the cost of ownership, an interface designed in-house along with a public domain implementation of SNMP V3 [4] will be used as an alternative to HP Open View.
- The GUI interface will be written in Tcl/Tk so that the interface can be rapidly prototyped and modified independently of the underlying capabilities.
- For the performance critical parts of the Management Console, the C programming language will be used.
- Standard agents will be used for devices such as the routers, and an agent-building package will be used for devices that require special management (SNAC cards).

- To further reduce the chances of concurrent actions affecting the timing of critical tasks, the Management Console will be implemented with several processes. This will allow the placement of a higher priority on the time critical functions of the SNMP session manager.

2.4.3- Management information

This subsection lists the managed objects and their required specifications.

- SNAC Cards (Common specifications)

SNAC card attributes:

- SNAC card ID, name, type
- Administrative state
- Operational state
- TCP/IP link status (MNG-SNAC card link status)
- Link Data rate
- Queue size
- Queue threshold
- Data compression (on/off)
- EMCON mode (normal, Rx, Tx)
- Priority Table
- Configuration files

SNAC Card Statistics:

- Data bytes queued during the last time interval
- Data bytes transmitted during the last time interval
- Data bytes successfully transmitted (ACKed) during the last time interval
- Data bytes retransmitted (not ACKed) during the last time interval
- Data bytes dropped during the last time interval
- Data bytes received during the last time interval
- Total data bytes queued
- Total data bytes transmitted
- Total data bytes successfully transmitted (ACKed)
- Total data bytes retransmitted (not ACKed)
- Total data bytes dropped
- Total data bytes received
- Link throughput expressed in bits per seconds (bps) and calculated as follows:

$$\text{Link throughput (bps)} = \frac{\text{Data Bytes Successfully Transmitted} * 8}{\text{Time interval (s)}}$$

Note: The time interval value will be predefined and is dependant on the SNAC card type. It should be implemented in a manner such that the throughput is calculated over the time interval that ends at the moment the measurement is taken (sliding window). For example, if the interval is defined as one minute, the throughput should be calculated using the Data Bytes Successfully Transmitted in the last minute.

- Link utilisation will be calculated as follows for ELOS:

At the end of each time interval,
If the transmit queue is not empty then (the link is always busy)
Link utilisation = 100%
Else

$$\text{Link utilisation (\%)} = \frac{\text{Data Bytes Transmitted} * 8}{\text{Time interval} * \text{link data rate}}$$

And calculated as follows for BLOS:

At the end of each time interval,

$$\text{Link utilisation (\%)} = 100 * \frac{\text{Data Bytes Transmitted}}{(\text{Data Bytes Transmitted} + \text{Fill Data Bytes})}$$

Where Fill Data Bytes is the number of bytes added to fill the message window

SNAC Card Actions:

- Reboot
- Flush queues

SNAC Card notifications:

- State change (device up/down)
- Connectivity (link up/down, quality good/poor)
- EMCON on/off
- Fault report
- Event report (e.g. message queue full/not full, queue above/below threshold etc.)

- SNAC Cards (Particular specifications)

HF BLOS

- Evaluation interval
- Number of channels
- Node's radio frequencies (Rx and Tx)

- Router (based on standard MIB definitions)

Router attributes:

- ID, Name, Type
- Interfaces
- Address translation
- IP routing table
- IP address table
- OSPF/MOSPF parameters

Router statistics:

- Interface statistics (number of bytes in, number of bytes out)
- IP statistics

3- High Level Design

This section presents the high level design of the network management solution proposed to satisfy the requirements listed in the previous sections. It is important to note that the design described in this section reflects only the current status of the prototype, and that the main focus of this design is the Local Management Console and the Agents. Potential solutions to issues specific to the Remote Management Application need to be further investigated.

The first subsection presents a general overview of the design. The second subsection presents the division of the functions in the Management Console. The third subsection presents the functions of the Agent. Finally, the fourth subsection presents issues that need to be addressed in any design proposed for the Remote Management Facility.

3.1- General design overview

This subsection presents a general overview of the proposed design. The design has elements in three distinct locations in the network: the Agents on the devices themselves, the Local Management Console on the LAN of the managed devices, and the Remote Management Application which can be located anywhere in the network. Furthermore, the decision to collect statistics at all times, even when the Management Console's GUI is not running, has resulted in a design where the software operates on two levels. The first level is the GUI, which is a user owned process that is started when the network manager logs in and halted when he or she quits. The second level encompasses daemon processes that are started at boot time and are always available to serve requests and to collect data. One must note that several functions at the daemon level are separated into different processes to ensure that the time-critical tasks for which they are responsible do not interfere with each other. Figure 3 presents an illustration of the proposed design. Each of the elements in this illustration is further described in the following subsections, with each subsection dedicated to a specific network location.

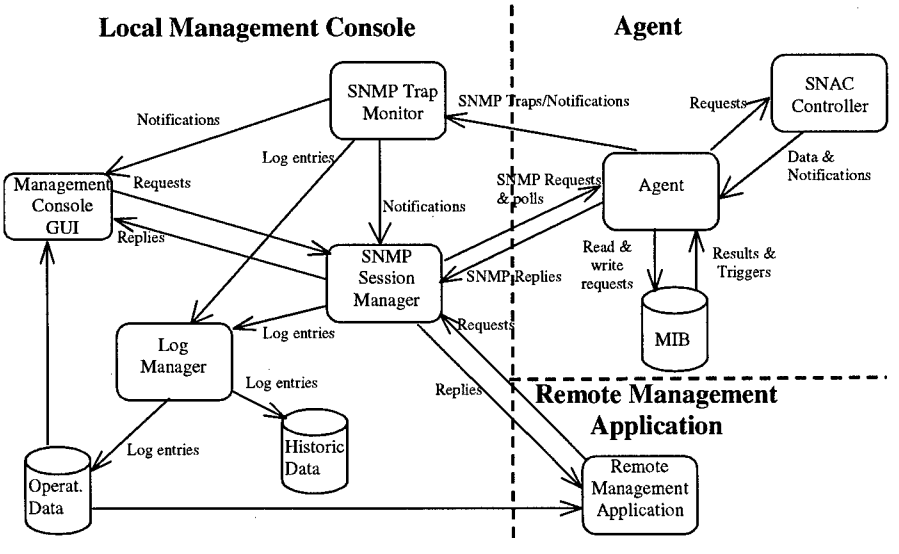


Figure 3 *Proposed design*

3.2- Local Management Console

This subsection presents the design elements of the Local Management Console. As shown in Figure 3, there are six main elements in the proposed design:

- Management Console GUI;
- Historic Data Repository;
- Operational Data Repository;
- Log Manager;
- SNMP Session Manager;
- SNMP Trap Monitor.

The **Management Console GUI** is the central means by which the user can control and access the network management capabilities of the system. The GUI will display status information of network devices and will permit the user to request more detailed information on specific devices. The GUI also allows the user to make network configuration changes and configure the SNMP Session Manager.

The **Data Repositories** are responsible for storing the collected data. This data will consist of two separate repositories: one for operational, or short term, data, which consists of data showing the recent status of the network, and one for historical data, which holds all data collected for later analysis. These data repositories are simple ASCII text files.

The **Log Manager** is responsible for the accumulation of status data for later analysis, and for managing the different files in the Operational Repository. The first of these responsibilities consists of polling the monitored devices, via the SNMP Session Manager, for specific MIB values at regular intervals, and writing the resulting data into the appropriate data repositories. The second of these responsibilities consists of removing outdated data from the Operational Repository at regular intervals. The main purpose of the Log Manager is to handle and minimise the overhead of physically writing status data to disk, without interfering with the time critical functionality of the SNMP Session Manager.

The **SNMP Session Manager** is responsible for monitoring the status of the network and for providing SNMP messaging services to other elements of the Management Console. The status monitoring capabilities consist of ensuring that messages are received from the monitored devices on a regular basis, and polling devices from which no messages were received during the last time interval. The messaging services of the SNMP Session Manager consist of encoding and decoding SNMP-V3 messages, and allowing other elements to send and receive these messages. In this role, the SNMP Session Manager forwards to the Log Manager all received requests so that they may be stored in the historic data repository. The Session Manager will use the public domain implementation of SNMP-V3 written for the NET-SNMP project[vi] [4].

The **SNMP Trap Monitor** is responsible for receiving SNMP Traps from the Agents and forwarding these notifications to the Log Manager for archival, and to the SNMP Session Manager and Management Console GUI for display to the user. This element will also use a public domain implementation of SNMP-V3 from Net-SNMP.

It is important to note that it is currently intended that the SNMP Session Manager, the SNMP Trap Monitor, and the Log Manager will be daemons that are run continuously once the Management Console is installed on a Workstation. This decision was made due to the design objective of having performance data collected at all times. As a result, the Management Console GUI is an application that runs only when an operator is logged in. The SNMP Session Manager and SNMP Trap Monitor send alarms to a given address whether or not a Management Console GUI is currently running.

[vi] NET-SNMP is a public domain implementation of SNMP that was originally based on the Carnegie Mellon University and University of California at Davis SNMP implementations.

3.3- SNMP Agents

This subsection presents the design elements of the device Agents. Each of the supported devices will have their own Agent. As shown in Figure 3, there are three main elements in the proposed design:

- SNAC Card Controller;
- Agent;
- Management Information Base (MIB).

The **Agent** is responsible for providing a management view of the system resources to the Management Console. It must validate and respond to management requests emanating from the Management Console, and issue notifications informing the Management Console of any changes that occur within the management view (such as values of specific variables that have changed, or the crossing of predetermined thresholds). In other words, the Agent is responsible for converting the physical view of the managed system into the appropriate management service, and for mapping a manager service request onto the correct physical resource. In addition to responding to configuration requests and forwarding spontaneous events to the manager, the Agent must also ensure that the values stored in the MIB are consistent with actual network resources. At a later stage, the Agents could be extended to include the ability to send the values of specific variables to the Management Console at periodic intervals without having received a request. This capability would be programmable by the Management Console through the addition of specific variables instances to the MIB.

The **Management Information Base** (MIB) is the management view of the actual resources under the Agent's control. The MIB is a complete collection of all managed objects, with their corresponding attributes, which are present within the managed system. Management operations being exchanged between the Management Console and the Agent reference the MIB. Agents must ensure the validity and integrity of the information within the MIB with which they interact.

Finally, the **SNAC Card Controller** is the core software component that provides the functionality of the SNAC card. In other words, it implements the link layer of the communication protocol for the low bandwidth link it controls. The SNAC Card Controller is the existing code base to which specific hooks will be added to allow the Agents to monitor and control the device.

3.4- Remote Management Application

This subsection describes the Remote Management Application. It is important to note that this discussion is only an attempt to illustrate the role that the Remote Management Application may play in the overall network management solution. The discussion also focuses on the issues that will need to be considered and resolved due to the additional complexity of Remote/Inter-node network management.

The **Remote Management Application** is a GUI that provides the means for the user to remotely control and access the network management capabilities of the proposed system. As such, the capabilities one would expect the Remote Management Application to provide are very similar to those provided by the Network Management Console GUI. The Remote Management Application GUI will display status information to the user, allow him or her to request detailed information on specific devices, and provide the means to make configuration changes to the network.

The main issue that needs to be addressed in Remote/Inter-node network management is the fact that data, including network management traffic, must travel over low bandwidth links. The capacity of these links can be rapidly exceeded, causing communication problems and possibly the failure of the network management function. Also, due to these low bandwidth links, the delay in obtaining desired information can become quite significant. This can make network management difficult due to inaccurate or outdated information.

Examples of short-term solutions to these difficulties could include allowing only limited network management to be performed remotely, as opposed to providing full remote monitoring and control.

Alternatively, the auto-discovery feature, which requires a lot of bandwidth, could be implemented with the use of static tables containing descriptions of all devices that form the ship's networks in the fleet. For example, all of a ship's devices and network configuration information could be added together either by loading a file upon request from an operator, or when the ship comes within communication range. The auto-discovery process could also be scheduled during off-peak hours to reduce its impact. However, all of these solutions impose limits on functionality that may not be acceptable in the long run.

Other potential solutions also focus on reducing the amount of traffic on low bandwidth links. For example, the operational data repository could be modified to keep only recent data, reducing the size of the logs that must be viewed to assess the current state of the network. Another example of a potential solution would be to handle alarms locally, filtering them before they are passed on to remote management sites. These solutions may well be acceptable and desirable in the long run. However, more data is required in order to propose a design for the long term. Rapid prototyping and data collection are necessary to obtain a better gauge of the processing, delay, and bandwidth cost of different solutions.

4- Agent Design and Implementation

This section presents the design and implementation of the SNAC's Agents, in particular, those for the ELOS and BLOS cards. It is important to note that the design and implementation described in this section is closely tied to the current implementation of the SNAC cards. These cards contain a lot of historic code, some of which is no longer necessary, but which has forced certain design decision on the current Agent implementation. For example, one historic feature of the code that is no longer being used is a sub-routing mechanism that allows the upper layers (SRIU) to route between several links. As of the time at which this document was produced, the implementation of the SNAC cards is being reviewed and the design of their Agents will need to be revised extensively if they are to be adapted to the ensuing version.

The first subsection presents a quick overview of the existing code in order to highlight areas where changes were required in the Agent. The second subsection contains a short discussion on the creation of the MIB definitions, and on the integration of the Agent with the existing SNAC card code. The third subsection presents the elements of the implementation that are common to both types of SNAC cards, namely the SRIU. The fourth subsection presents implementation details specific to the ELOS SNAC card. Finally, the fifth subsection presents implementation details specific to the BLOS SNAC card.

4.1- Generic SNAC Card Agent design

This subsection presents a brief history of the code used in the ELOS and BLOS cards as well as a broad overview of where changes are required in order to add SNMP Agents to the existing code. The remaining subsections will further detail the changes in each of the development areas outlined below.

Historically, the SNAC cards were part of a VME card cage assembly comprising several SNAC cards and one SRIU card, all of which were running the VxWorks™ operating system. The SRIU card was responsible for controlling the SNAC cards, as well as for routing the outbound traffic using a route cost algorithm. The SRIU card was also responsible for handling the Ethernet link with the router, as all inbound traffic passed through the SRIU. Each SNAC card was responsible for reliably delivering messages over the low bandwidth link it controlled. As a result of the hardware iterations being developed at the same time as this project, each SNAC card now includes its own SRIU. In other words, the code of both the SRIU and the SNAC has been modified to co-exist on the same card, and use the same processor and memory space. Nevertheless, the SNAC and the SRIU still have separate code bases; meaning that the SRIU and SNAC each still have their own distinct sub-set of processes.

The fact that both the SNAC and the SRIU are now on a single card implies that the SRIU no longer has routing capabilities, as it is connected to only one link. The SRIU now receives and forwards to the SNAC only the messages to be sent over the link with which it is associated. There are also several messages exchanged between the SRIU and the SNAC it is controlling. Of particular interest is the Queue Report message, which is sent to inform the SRIU of any state changes in the message queues. This message is used to inform the SRIU when the SNAC is being overloaded with messages so that the SRIU can adjust its routing parameters. Also important to the design is the fact that the SNAC contains time critical behaviour that should not be needlessly hampered by management functions. All of these details brought us to the conclusion that the Agent software should be incorporated within the SRIU, as the SRIU code contains some control features over the SNAC, and it also contains state information about the SNAC.

The first requirement for the Agent design is the capability to produce recurring statistics on the performance of the link. By their nature, these statistics can only be generated in the SNAC code as they are based on events such as messages being retransmitted or dropped. These events are dependant on the inner workings of the link layer and remain hidden from the SRIU. Handling these statistics requires functions to calculate their values and a mechanism to pass them on to the SRIU so they are available to the agent when needed.

The second requirement for the Agent design is to generate SNMP Traps for specific events. Events can be split into two categories: status Traps and performance Traps. The SRIU will generate status Traps as it is responsible for managing the SNAC and therefore holds the necessary general status information. The SRIU will also be responsible for generating the performance Traps as it also possesses enough information to determine when they are necessary.

In summary, changes to the Agents were required in seven areas of code:

- The definition of the MIB and the interface between the chosen Agent and the SRIU;
- The mechanism at the link level (SNAC) to calculate performance statistics;
- The definition of the Statistics Report message in both the SRIU and SNAC;
- The SNAC code generating the Statistics Report message;
- The SRIU code handling the Statistics Report message;
- The SRIU code handling the Queue Report message;
- The SRIU code routing a message to a SNAC.

4.2- MIB definition and Agent's Incorporation

This subsection presents the origin of the definitions that were added to the standard MIB-II in order to monitor the SNAC card Agent's behaviour, as well as how these definitions are incorporated into an Agent that interacts with the SRIU.

The variable definitions that were added for the Agent are loosely based on the MIB definitions used for an old ELOS Agent created by the US Advanced Data Network System team in VxWorks 5.2 with the help of VxWork's optional SNMP package. It is important to note that this package was not used for the current implementation in VxWorks 5.4 as it did not support SNMP Version 3. The file containing the MIB variable definitions extending the standard MIB-II is attached in "Annex B; Supplemental MIB Variables definition file".

RFC 1155 [1] specifies that each MIB variable must be uniquely assigned a leaf in the MIB tree to allow its unique identification. This RFC also specifies the precise places in the MIB tree where variables can be added. Just as in the MIB inherited from the US team, the new definitions were added under the navy's experimental private enterprises sub-tree (ID 1.3.6.1.4.1.1738); more precisely, in the sub-tree where the MIB variables used for the Advanced Data Network System's CRIU project, from which the SNAC code is derived are defined (ID 1.3.6.1.4.1.1738.2.300).

The inherited definitions were divided into six sub-groups: the administrative functions of the upper level functions (criuAdmin), the administrative functions of the lower level functions (capAdmin), the general statistics (capStats), the per-application statistics (appstats), the Traps (criuTraps), and the application priority table (priority). As it was required to implement a new set of general statistics and to have specific SNMP Traps while preserving the other already implemented behaviour, the variables were kept in the same sub-tree with two exceptions. The first being that the definitions under the general statistics and Traps sub-trees for the variables described in section "2.4.3-Management information" were completely replaced. The second being that all indexing by SNAC IP address was removed from the original MIB-II extension file since, in complementary work, the upper level functions (SRIU) were modified to control only a single link, and all indexing was removed from the source code.

One of the first steps required, from the developer's point of view, is the choice of the Agent and of the SNMP protocol stack implementation. Given that such an implementation requires a large amount of work, and that there are several existing source code implementations available, it was decided that a pre-existing implementation would be used. The selected implementation is Emanate/Light Agent from SNMP Research International, Incorporated™. This implementation was chosen both because a version for VxWorks was available, and because it supports SNMP V3.

On VxWorks, the Emanate/Light Agent is coded in such a way as to run as a separate process that communicates with other processes through a specific set of interface functions. Stubs for those functions

are created when the MIB is compiled with the Mosy/PostMosy tools. It is important to note that in this architecture the responsibility for keeping the actual values of the MIB variables is given to the implementation of the interface functions. Since most of the required values are already in the existing code, and in VxWorks all global variables are visible across all processes, implementing these functions simply requires that the values of the internal variables be copied to the data structures of the functions.

All of the Agent customisation source files are located in the "mib" subdirectory of the TD project source code directory. There are two files of particular interest: the first, "elos.my", contains the definitions of the added MIB variables, and the second, "k_elos.fin", contains the definitions of the interface functions. The other two files in the "mib" subdirectory are make files that will compile either an SNMP V1 or V3 Agent. In order to incorporate this Agent into the running SNAC you need only to add two lines in the card's start up script. The first of these lines assigns to the SR_AGT_CONF_DIR environment variable the location of the Agent's configuration file, and the second starts the Agent's process. For example, on the author's machine, these two lines are:

```
Putenv("SR_AGT_CONF_DIR=/opt/packages/Henryk/TD/conf")
Sp SNMPD_main
```

The table below summarises the different files and their location in the Agent code subdirectories.

Description	Source Code file	Header file	File location
Defines the variables added to the MIB-II	elos.my	-	mib
Defines the functions forming the interface between the Agent and the SNAC for the added MIB variables.	k_elos.fin	-	mib

Table 1 *Location of the files containing MIB code changes*

4.3- SRIU Implementation

This subsection presents in more detail the changes made to the SRIU code. As we have seen in Section "4.1-Generic SNAC Card Agent design" development was required in four areas in the SRIU code. This subsection presents each of these areas in turn.

The first identified area was the definition of a Statistics Report message. By analysing the code, it was discovered that the SNAC was already sending some cumulative statistics at regular intervals to the SRIU through a report (STATISTIC_REPORT_TYPE). The statistics received through this report are kept in the SRIU along with SNAC status information. It was decided that the same mechanism would be used to send and keep the new statistics so that they would be available when needed by the Agent. The format of the Statistics Report message is defined in a simple data structure that is contained in the header file "snac.h". Several variables were added to the "STATISTICS_REPORT" structure, one for each of the new performance statistics. It is important to note that this data structure is mirrored in the ELOS and BLOS code in the header file "snac_int.h" of each platform, and that all of these files must be kept in sync.

The second identified area was the section of code that handles the Statistics Report messages. Two actions are required to handle this message type. First of all, a check is made to determine whether the ratio of retransmitted bytes versus transmitted bytes indicates that the quality of the link has passed a predefined limit. If this is the case, the link status is stored in a new variable (poor_link_flag), and an appropriate call is made to generate an SNMP Trap (CRIUDoTrapSend). This ratio is checked by a call to a

new procedure: "check_link_quality". Another change made related to the handling of the Statistics Report messages is that all the new statistics added in the messages are copied in new variables in the internal state data structure of the SNAC (struct snac_circuit_cb). These values are stored for use by the Agent in response to SNMP requests. The values are stored by the new procedure: "save_stat". Both of these changes were made in the module "metric.c".

The third identified area was the code that sends messages to the SNAC to be transmitted over the link it controls. When such a request is made a performance Trap may be generated, since the SNAC will block its message channel when its highest priority queue is full. This causes the message forwarding function in the SRIU to fail. The need to send a "snacMessageQueueFull" Trap is therefore detected when an attempt to delegate to the SNAC the sending of a new message fails because the conduit is blocked and the previous known state was that is was not blocked. Vice-versa, the need to send the "snacMessageQueueNotFull" Trap is detected when an attempt to send a new message succeeds and the previous known state was that the conduit was blocked. When either of these conditions is met a call is made to generate the appropriate SNMP Trap (CRIUDoTrapSend). This is handled by the procedure "sendToSnac" in the module "mimsnacInterface.c".

Finally, the fourth identified area was the code handling the Queue Report message. This report is sent to the SRIU by the SNAC when the number of bytes in one of its transmit queues crosses a certain threshold. This threshold is defined as being any multiple of a certain boot parameter. More specifically, the appropriate Trap is sent only if the threshold crossed is one where the multiple of the boot parameter is equal to one. In this instance the SRIU must determine, by examining the value of message attributes, wheather a "snacQueueAboveThreshold", "snacQueueBelowThreshold" or "snacMessageQueueNotFull" Trap should be generated. For example, when the previous known state of the message channel to the SRIU was blocked and a Queue Report is received, the "snacMessageQueueNotFull" Trap is generated, since this report indicates that the message queue is above the threshold but not full. The Queue Report message is handled in the procedure "input_snac_circuit" of module "mimsnacInterface.c".

Some other minor changes to the code have been made. A call to generate the "SNAC up" Trap was added to the end of the initialisation phase of the SRIU. A call to generate the EMCON or the "SNAC down" Traps was added to the code that handles changes in the values of the corresponding MIB variables. Two other status Traps, "link up" and "link down" are not yet implemented since, by design, there is no existing mechanism for the SNAC to inform the SRIU of those events.

All of the new status variables for the Traps as well as the new statistics being calculated have been added to the data structure "snac_circuit_cb" which is defined in the module "includes.h". Furthermore, all of these variables are initialised in the function "init_snac_circuit". All of the performance variables are reset in the function "init_circuit_history", which implements the reset statistics function of the Agent (activated by the MIB variable "capStatisticReset" or 1.3.6.1.4.1.1738.2.300.2.10). Both of these functions are defined in the module "mimSnacInterface.c". The different files and their location in the SRIU code subdirectories are summarised in Table 2.

Description	Source Code file	Header file	File location
Defines the format of statistics report messages	-	snac.h	Sriu
Manages statistics related to communication with the SNAC and registered clients	metric.c	-	Sriu
Manages the routing algorithm used between SNACs	mimSnacInterface.c	-	Sriu

Table 2 *Location of the files where SRIU logic has been changed*

4.4- ELOS Implementation

This section presents in more detail the changes to the ELOS code required to support the new Agent. As described in section "4.1-Generic SNAC Card Agent design", development was required in the following three areas: the mechanism to calculate the performance statistics, the definition in the ELOS code of the Statistics Report message format, and the code generating the Statistics Report message. This section presents each of these areas in turn.

Calculation of the new statistics required two separate development efforts. The first was to create functions that would manage the moving time slot for the "instantaneous" statistics, calculate them, increment the cumulative statistics, and provide access to the different statistics. The second was to identify the places in the code where the events being tallied are managed and add the appropriate calls to the functions previously defined.

From the onset, it was decided that the moving time slots would be managed using linked lists where each element of a list holds the value and time of an individual reading. Each list is used to hold the elements in chronological order, and any element older than the time slot duration is removed whenever the list is accessed. Several functions were created for these lists: a function to create a list and set its time slot duration ("list_create"), a function to add readings ("list_add"), and functions to calculate various statistics from the lists ("list_sum", "list_average", and "list_rate"). These functions are located in the file "list.c" and they all control access to the lists by other processes with the use of semaphores.

For every event upon which new statistics are based, a function is created to calculate the corresponding statistics, and most of them have a similar implementation. For a list of these events, please refer to Table 3. In each of these functions, the new value passed through the parameter is added to the variable that contains a total of the received readings, and an element is added to a corresponding moving time slot list instance. An exception to this general implementation is the function used to process the PDU "queued" and "dequeued" events. The function used to process these events is different because the corresponding link utilisation statistic does not have an event per say but instead needs to know the number of PDU in the processing queues. More specifically, the link utilisation is based on a function that checks once a second whether there are any PDU in the queues. Should there be one or more PDU in a queue this would indicate that the link is being utilised. A separate process was created to make these readings, which uses a variable that tracks the presence or absence of any PDU in the processing queues. This variable is incremented in the queued event function and decremented in the dequeued event function. Each of the readings is then added to a time slot list that is used to calculate the link utilisation. Note that the "dequeued" event's sole purpose is to track the number of PDU in the processing queues, and consequently its corresponding function does not store a total or a time slot list. Another exception to the general statistic function implementation is the function that calculates the link throughput statistic. This statistic is based on the current number of acknowledged bytes, and is calculated using the time slot list for the number of acknowledged bytes. Access to all of these statistics is provided by dedicated functions, all of which are contained in the file "stats.c".

Event	Definitions
Queued	A PDU was added to the priority queues for transmission
Dequeued	A PDU was removed from the priority queues.
Transmitted	A PDU was written to the hardware buffer of bytes to transmit over the link
ACKED	An acknowledgement was received for a PDU.
Not ACKED	A PDU was transmitted but it did not receive an acknowledgement during the allowed time.
Dropped	A PDU was dropped from the priority queues. This can occur because its Time To Live (TTL) expired before it could be transmitted, or because another error, such as a full priority queue, caused the packet to be dropped.
Received	A PDU was correctly received.

Table 3 *Events upon which statistics are based*

The part of the mechanism to calculate the new statistics that required the most effort to code is the insertion of calls to the functions that tally the statistics at the appropriate places. This requires intimate knowledge of the code implementing the ELOS SNAC. Investigation of the code has revealed that the most appropriate places to insert these function calls are in the module functions managing the transmit queues. When messages to be sent are received by the ELOS SNAC they are inserted, according to their priority, in the appropriate transmit queue. The function used to make the final message validity checks and insert the messages is "XQ_X_AddXmitDg". It is in this function that the call to the function that tallies queued bytes is inserted. Similarly, the function used to remove a message from the queue is "XQ_X_DeleteXmitDg", and the call to tally the messages removed from the priority queues ("dequeued") is inserted in this function. It is important to note that the calculation of how many messages are on the SNAC using the aforementioned "queued" and "dequeued" calls is valid only because messages are added or removed from the SNAC only with corresponding calls to these two functions. The calls to tally transmitted bytes are inserted at the time when the SNAC builds the buffer that is to be written to the serial port. This buffer is created within two procedures: "XQ_GetLoc_Msgs" and "XQ_GetXmitd_Msgs". The former adds new messages to the buffer, and the latter adds messages to be retransmitted to the buffer. The call to tally retransmitted bytes is also put in the "XQ_GetXmitd_Msgs" function; in fact, bytes added to the buffer by this function are always tallied twice, once by the transmitted bytes statistic, and once by the retransmitted bytes statistic. The call to tally ACKED bytes is added to the "XQ_ProcessAckTask" function. Finally, the call to tally the dropped bytes is added in the "XQ_CullXmitQ" function, whose purpose is to scan all of the transmit queues and to remove the messages once their TTL has expired.

Changes in the two remaining areas are straightforward. As has already been stated in Section "4.1-Generic SNAC Card Agent design", the format of the Statistics Report message, as defined by the data structure "STATISTICS_REPORT" which is located in file "snac_int.h", needs to mirror any changes to the format of this structure in the SRIU. The Statistics Report is generated in the Statistics Managing task's main loop (function ST_Mainloop) when it receives from the SRIU either a "GS_INTERVAL_STAT_REPORT" or a GS_SNAPSHOT_STAT_REPORT" request. In each case, a call to a new function is inserted to fill in the new fields. This function, "NewStats", calls the access function for each of the added statistics, and saves the result in the appropriate spot in the Statistics Report message. Table 4 lists the files that have been changed and gives their location in the ELOS code subdirectories.

Description	Source Code file	Header file	File location
Generic code for managing interval lists.	list.c	list.h	elos/st
Basic arithmetic operations on time intervals.	timeval.c	timeval.h	elos/st
Cumulate, manage, and provide access to each of the new statistics.	stats.c	stats.h	elos/st
Define the format of the Statistics Report message.	-	snac_int.h	elos/gs
Routines that handle transmit queues.	code_xq.c code_xq1.c	-	elos/xq
Generate the statistics report message.	code_st.c	-	elos/st

Table 4 *Location of files where ELOS logic was changed*

4.5- BLOS Implementation

This subsection presents in more detail the BLOS code changes required to support the new Agent. As discussed in section "4.1-Generic SNAC Card Agent design", development was required in the following three areas: the mechanism to calculate the performance statistics, the definition of the Statistics Report message format in the BLOS code, and the code generating the Statistics Report message. This section presents each of these areas in turn.

In the BLOS, the calculation of the new statistics follows the same pattern as in the ELOS case. It is thus not warranted to dwell further on the functions that manage the moving time slot for the "instantaneous" statistics, calculate and increment the cumulative statistics, and provide access to the different statistics. The main difference between the ELOS and the BLOS is that in the BLOS an internal statistic, "fill bytes", is used to calculate the link utilisation as opposed to the process used in the ELOS of regularly checking whether the queue is empty. The internal BLOS statistic used to calculate link utilisation tracks the number of filler bytes that are transmitted to complete time slots when there is not enough data to transmit. One further difference between the ELOS and the BLOS is that the transmitted and retransmitted (not ACKED) statistics of the BLOS do not tally the size of the actual data, but instead track the number of bytes physically sent on the wire. These statistics must be calculated in this way since the data on the BLOS is transmitted in a compressed form and there is no way to know the size of the real data at the level where these two statistics can be cumulated. The other statistics, (Queued, ACKED, Dropped and Received), tally the actual size of the data. Regardless of the differences in the meaning of these statistics between the ELOS and the BLOS versions (due to the transmission compression scheme), the same MIB variable names are used.

Once again, the part of the mechanism used to calculate the new statistics that required the most effort to code is the insertion of calls to the functions that tally the statistics in the appropriate places. Because the code of the BLOS is very different from that of the ELOS the places in the code where the events being tallied are managed are also very different. This means that intimate knowledge of the code implementing the BLOS SNAC is required in order to add the appropriate function calls. The discussion in the following paragraphs does not intend to present the architecture of BLOS; it only aims to present enough of it so that the insertion points may be understood.

The first calls are inserted at the time when the BLOS receives a request to send a message over the link. This request originates in the process started by the TCP server of the SNAC to handle the connection with the SRIU. The request eventually reaches the procedure "insertQElement" which adds the PDU to an appropriate priority queue. This involves a check to verify whether there is enough space in the

given queue for the PDU. If enough space exists, the PDU is inserted and it is added to the tally of queued PDU through a call to "Stats_PDU_Queued". If there is not enough room in the queue the PDU is dropped, and it is added to the tally of dropped PDU through a call to "stats_PDU_Dropped". The priority queues are accessed by three different processes: the command execution process for the "flush queues" command, the process that regularly monitors the queues for statistical purposes, and the process that selects the highest priority PDU, compresses it, and sends it on to the next processing step. All of these processes access the queues through the procedure "accessTXPDUinQHead" which returns the oldest PDU in the queue that has not expired. This procedure also drops any expired PDU it finds in the queue. Any PDU that is dropped is added to the tally of dropped PDU through a call to "Stats_PDU_Dropped". It is important to note that the uncompressed size of the message is now stored in a new variable "init_len" in the packet management data structure ("packetinfo").

The next step in processing a request is the "Segment" task, which divides each message into data frames and forwards them to the "Transmit" task. The "Segment" task is also responsible for sending frames that were not ACKED to the "Transmit" task for retransmission when the packet's timer expires. It is here that the retransmitted bytes are cumulated through a call to the procedure "Stats_PDU_Not_Acked".

The "Transmit" task is responsible for the actual write to the serial port, and it is here that the transmitted bytes are cumulated through a call to "Stats_PDU_Transmitted". This task must also ensure that the connection with the remote device remains active, so, if needs be, it will add filler bytes to the port. These filler bytes are cumulated through a call to "Stats_PDU_PhilBytes". When the last frame of a packet has been sent the "Transmit" task starts a timer by sending a message to the timer task.

The timer task is responsible for the management of the timers. It does so by maintaining a list of pending timers ordered by their expiry time. The timer task can receive three different messages: start timer, stop timer, or timer expired. When a "start timer" message is received, the request is queued and the first timeout is calculated. When a "timer expired" message is received the timer queue is searched for all timers that have expired and a message is sent to the "Segment" task for each of these timers so that the required retransmissions are made. The "timer expired" messages are generated when a call to "mseQReceived" returns with no data because the calculated time out has expired before a message was received. Finally, when a "stop timer" message is received, the corresponding timer is removed from the list and its attached packet is removed from the active packets list. It is at this point that ACKED messages are tallied with a call to "Stats_PDU_Acked".

The receive message task (RxTask) is the task that reads incoming messages on the serial port of the link controlled by the SNAC. This task recognises ACK messages and reports them by sending a "stop timer" message to the timer task. It will also send data frames to the "Desegmenter" task. The "Desegmenter" task's responsibility is to reassemble the different frames and forward the results to the "Link2Client" task. It is the "Link2Client" task that finally forwards the packets to the recipient. This is implemented in the procedure "GetFromLinkToClient" and it is in this procedure that the last call to tally a statistic, bytes received, is inserted ("Stats_PDU_Received").

The changes made in the remaining two areas are straightforward. As was stated in section "4.3-SRIU Implementation", the format of the Statistics Report message in the data structure "STATISTICS_REPORT", which is defined in the file "snac_int.h", needs to mirror any changes made to this structure in the SRIU. Statistics Reports can be generated either for the local console or for a remote client reached via the SRIU. The same data structure is used in all cases, and its values are filled using the same function. The values for the Statistics Report message are accessed and stored in the appropriate spot in the data structure using the "ServerGetClientStatistics" function found in the file "snac_server.c". Appropriate statements for each of the new statistics were added to the "ServerGetClientStatistics" function to fill in the corresponding spots in the Statistics Report message. These statements make use of the access function defined in the "stats.c" file. The files that have been changed and their location in the BLOS code subdirectories are summarised in Table 5.

Description	Source Code file	Header file	File location
Generic code for managing interval lists.	list.c	list.h	Code: blos_pt161/stats Header: blos_pt161/h
Basic arithmetic operations on time intervals.	timeval.c	timeval.h	Code: blos_pt161/timeval Header: blos_pt161/h
Cumulate statistics, manage the interval lists, and provide access to each of the new statistics.	stats.c	stats.h	Code: blos_pt161/stats Header: blos_pt161/h
Define the format of the Statistics Report message.	-	snac_int.h	blos_pt161/h
Interface with SRIU (TCP server); this file also generates the different versions of the Statistics Reports.	snac_server.c	-	blos_pt161/net
Implementation of the network level for incoming messages.	Net_tx.c	-	blos_pt161/net
Management of the lower level queues and the physical link.	link2.c	-	blos_pt161/blos2

Table 5 *Location of the files where BLOS logic was changed*

5- Management Console design and Implementation

This section presents the design and implementation of the management console. The first subsection presents the messages exchanged between the different elements. The second subsection presents the design and implementation of the SNMP Session Manager. The third subsection presents the design and implementations of the Management Console's GUI. The fourth and fifth subsections propose possible designs for the two elements in the high level design that have not yet been implemented. These last two subsections therefore contain suggestions for further work or development. The fourth subsection presents a short description of the intended implementation of the Log Manager, and the fifth subsection presents a short description of the intended implementation of the SNMP Trap Monitor.

5.1- Inter-process communications

Due to the decision that the software on the Management Console would run at two levels, the design of a communication protocol between its various processes was required. This section presents the inter-process communication scheme that is used between the different elements of the Management Console. Several issues were considered during the design of the Management Console. Each of the major communication issues is presented in this section in order to explain how the solution was reached.

Each interaction between the various elements of the Management Console requires an inter-process communication protocol. The first such interaction occurs between the GUI and the SNMP Session Manager to allow the latter to serve the SNMP requests made by the user. Another interaction occurs between the GUI and the Log Manager in order for the GUI to access historic data. Since the GUI process is not created at the same time as the other processes, it needs a static way to address both the Log Manager and the Session Manager. It was decided that each server would be reached through a socket with a fixed Internet Protocol address and port (AF_INET). The SNMP Session Manager will listen on port 5555 and the Log Manager will listen on port 5556. In both instances, the GUI does not need to listen on a particular port since the two managers simply use the sender's address to return the results of any requests. However, the SNMP Trap Monitor needs to talk to all of the other aforementioned processes in order to notify them of important events, and the GUI thus also listens to a defined port, port number 5557. The SNMP Session Manager also needs to send log data to the Log Manager, but the mechanism described above is sufficient for this purpose, so the same port will be used. Figure 4 illustrates the communications paths between the processes on the Management Console.

In designing the format of the messages exchanged between each of the processes the first point to consider is the fact that the GUI is written with Tcl/Tk in which every object is a string. To simplify communication between the processes and to simplify debugging, it was decided that messages would be formatted in readable text. However, since only the SNMP session manager and the GUI are implemented, only the messages exchanged between the GUI and the SNMP Session Manager have been defined. The format of the commands the SNMP Session Manager can receive are all in the following pattern (where names between < > represents symbols that must be replaced by appropriate values): <command> <Target IP address> <space separated list of attributes>. Table 6 summarises the syntax of the commands that the GUI can send the SNMP Session Manager. In the syntax used, square brackets delimit a replacement value, "|" represents a choice between the values it separates in the square brackets, "1+" means that one or more instances of the preceding value is required and "0..1" means that the preceding value is optional but cannot be repeated.

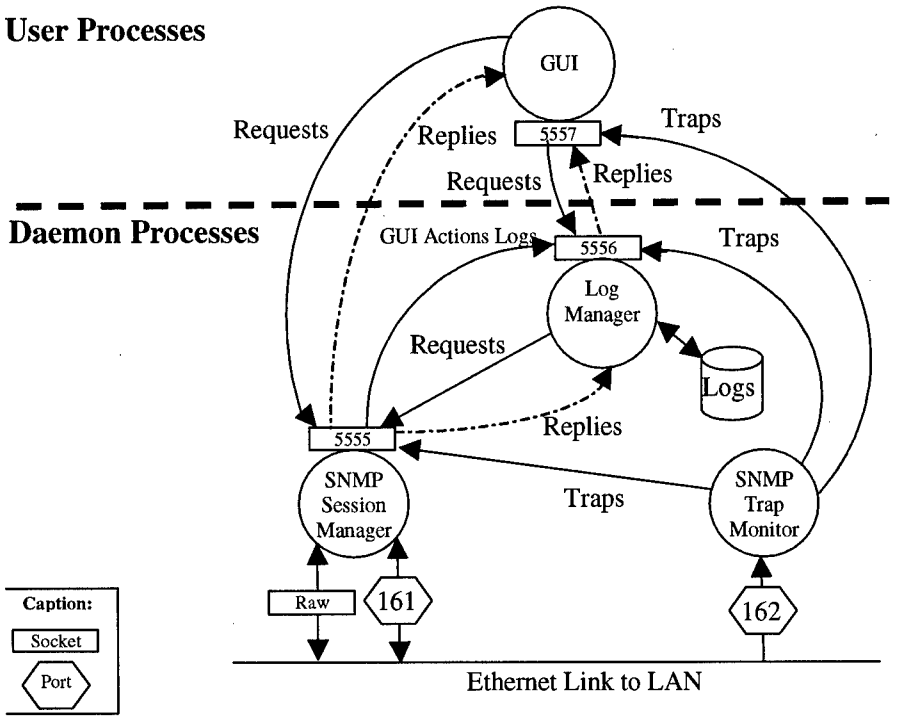


Figure 4 *Management Console Inter-process communication*

<message content> =	[<Ping> | <Get> | <Set> | <Getnext> | <Walk>]
<Ping> =	[ping <Device IP> <Ping attributes>]
<Get> =	[get <Device IP> <Get attributes>]
<Set> =	[set <Device IP> <Set attributes>]
<Get next> =	[getnext <Device IP> <Get attributes>]
<Walk> =	[walk <Device IP> <Variable name>]
<Ping attributes> =	[<Nb Ping > | <Nb Ping> <Intermediate message option>]$^{(0..1)}$
<Nb Ping > =	<Integer greater than 0>
<Intermediate message option> =	[y | Y | n | N]
<Get attributes> =	[<Variable name>]$^{(1+)}$
<Set attributes> =	[<Variable name> <value> <type>]$^{(1+)}$
<Device IP> =	[<IP Version 4 address of target device>]
<Variable name> =	[<OID> | <MIB identifier>]
<type> =	[i | u | t | a | o | s | x | d | n | I | F | D] Where i for integer u for unsigned t for time ticks a for IP Address o for object ID s for string x for hexadecimal number d for ASCII numbers n for Null value I for Integer (64 bits) F for Float D for Double float

Table 6 *Command message syntax of the SNMP Session Manager*

The replies that SNMP will send back follow a similar pattern: <command return code> <Device IP> <response attributes>. Table 7 summarises the syntax of the responses returned by the SNMP Session Manager.

<Reply message> =	[<Ping> | <Get> | <Set> | <Getnext> | <Walk>]
<Ping> =	[<Ping return code><Device IP><Ping X results>]
<Get> =	[<get return code><Device IP>< X results>]
<Set> =	[<set return code><Device IP> < X results>]
<Get next> =	[<getnext return code><Device IP> < X results>]
<Walk> =	[<walk return code><Device IP> <Walk X results>]
<Ping return code> =	[ping_intermediate | ping_timeout | ping_summary]
<Ping summary results> =	[<nb_ms_sent> <nb_ms_rec> <Average nb_sec> <average nb_usec>]
<Ping intermediate results> =	[<nb_ms_sent> <seq_nbr> <nb_sec> <nb_usec>]
<Ping timeout results> =	[<nb_ms_sent> <seq_nbr>]
<get return code> =	[get_answer | get_failure | get_timeout]
<set return code> =	[set_answer | set_failure | set_timeout]
<getnext return code> =	[getnext_answer | getnext_failure| getnext_timeout]
<Xresults> -	[<SNMP answer> | <SNMP failure> | <SNMP timeout>]
<SNMP answer> =	[<OID> <value>]
<SNMP failure> =	[<error message>]
<SNMP timeout> =	
<walk return code> =	[walk_answer | walk_failure | walk_timeout | walk_end]
<walk X results> =	[<X results>]
<Device IP> =	[<IP Version 4 address of target device>]
<Variable name> =	[<OID> | <MIB identifier>]

Table 7 *Reply message syntax of the SNMP Session Manager*

5.2- Design and implementation of the SNMP Session Manager

This section presents the design and implementation of the SNMP Session Manager, which is the most time critical element of the Management Console. As such, the purpose of the SNMP Session Manager is to provide access to SNMP messaging services, in order to avoid delays that may be caused by other components, such as the GUI's window management, or the Log Manager's disk accesses.

The Session Manager is divided into several functional modules each with their own responsibilities. Each of these modules is coded using header files (.h) and implementation files (.c), which allow access to their functionality only through a specific set of functions, and which hide the internal declarations of variables and data structures. The main modules are:

- The main code, which consists of the main event loop; it is responsible for the process's initialisation, waits for events such as timeouts or messages, and completes the initial dispatching of received messages;
- The inter-process communication translator, which is responsible for translating into specific actions the requests received from the Management Console's other processes;
- The session manager, which is responsible for managing SNMP sessions, and for calls to the API set of NET-SNMP.
- The time out queue, which provides timer management services to other modules;

- The ping module, which handles ping requests originating internally from the SNMP Session Manager, and from other processes, notably the GUI;
- The NET-SNMP API set, which is used to format and manage the UDP protocol stack for SNMP messages.

The function "main" of the SNMP Session Manager initialisatises the other modules used in this process, and waits for either a message on one of several ports or for the end of a calculated time-out period. The main building block of this functionality is the UNIX "select" function. This function waits for a limited amount of time for a message on any one of a specified set of sockets. Primitive functions from the time out queue as well as the NET-SNMP API set are used to calculate the maximum time the select statement will wait for a message. Once a message is received, a primitive from either the ping facility, the IPC translator, or the NET-SNMP API set, depending on the socket from which the message came, is called to read and process it. If the select statement returns without having received any message other primitives are called in the timeout queue and the NET-SNMP API to handle the time-outs that have expired.

The inter-process communication translator's facilities are used through two functions. The first one, "open_session_mgr_socket", is called during initialisation in order to open a socket on port 5555. The second function, "receive_command", is called each time a new message is received on the socket to handle the received command. The latter function reads the message from the buffer, saves the address to which the results should be sent, and calls "handle_GUI_request" to decode the command and execute the instructions it contains. Each command type is processed using a pair of functions that answer the given request. The first function decodes the command parameters specific to it, and calls the function from the module that will serve the request. The second function is a "call back" procedure that is called by the serving module when the results for the request are ready, or in the case of the walk or ping commands, partially ready, and forwards these results to the requesting process. For example, the Get command is served by the "handle_get_request" and "SNMP_get_call_back" functions.

The session manager is an interface that handles communication sessions with the Agents. A session is a set of protocol parameters used to facilitate message exchange with a specific device. The session manager has several functions, centralising a set of behaviour into a single point of implementation. Its first function is to hide the complexity of the underlying API set from NET-SNMP. Its second function is to ensure that only one SNMP request has been forwarded to a specific Agent regardless of who originated the request. Any open session with a device is shared between all the requestors and concurrent requests are queued in a FIFO queue. Finally, the session manager is responsible for pinging the device upon the failure of a request to ensure that the device can still be reached. If the device answers the ping, the Session Manager will move on to the next request in the queue, if not, it will flush the request queue and send an error to the sources of all of the flushed requests. The session manager will serve all SNMP requests through three functions. The first function, "open_device_agent_session", allows its caller to initiate a session with a device by opening an SNMP session and setting the parameters defining the communication protocol to be used in the dialog that follows. This function returns a key that identifies the session during further interactions. The second function, "send_device_agent_pdu" is used to send a PDU (the content of an SNMP message) to the remote device using the session identified by the aforementioned key. This function has a parameter that allows the caller to identify the functions that should be called back when the reply from the remote device is received. Finally, the third function, "close_device_agent_session", allows the caller to close a session that it had previously opened.

The time out queue is a facility that maintains a list of time out events ordered by their expiration time. The time out queue interface functions allow adding (add_timeout_event), rescheduling (reschedule_timeout_event), and cancelling (remove_timeout_event), a time out event. The "add_timeout_event" function has a parameter that specifies the function that should be called back when the added timeout expires and returns a key, which identifies the time out event, which may be used to reschedule or cancel the time out. The time out queue is implemented to work in conjunction with a UNIX select statement whose invocation is the responsibility of the function using the time out queue facility. Consequently, the "get_first_timeout_delay" function is used to obtain the time remaining before the first time out in the list will expire in accordance with the maximum waiting time parameter of the select

statement. Furthermore, in order for the time out queue to function correctly, the procedure "check_timeout_events" must be called on a regular basis, especially when the select statement returns because the maximum waiting time specified in its invocation has expired.

The ping module permits the calling code to send ICMP ECHO messages to any IP address and interprets the received messages. The program using the services offered by this module, in this case the function "main" of the SNMP Session Manager, must perform three tasks. Firstly, it must call the "init_ping_facility" function during initialisation with super user privileges so that a "raw socket" can be opened. This function returns the socket ID of the opened socket. Secondly, in the processing loop, it must use this socket ID in a select statement and call the "receive_v4_ping" function when data is received on this socket. Finally, it must initialise and use the time out queue facility described above. Once these tasks are integrated into the main code, pings can be requested with a call to either one of the following functions: "add_ping_request_with_address", or "add_ping_request". These two functions contain a parameter that specifies a call back function to be used when an echo message is received, or the request times out.

Table 8 summarises the relevant files and their location in the code subdirectories. Figure 5 illustrates the code dependencies between the modules that have just been described.

Description	**Source Code file**	**Header file**	**File location**
Main loop and initialisation of the SNMP session manager.	serveur_main.c	-	manager/serveur
Command translation and implementation.	serveur_IPC.c	serveur_IPC.h	manager/serveur
Session manager and API calls.	session_manager.c	session_manager.h	manager/serveur
Ping facility.	pinger.c	Pinger.h	manager/serveur
Command data structure definition and functions used to manage command queues.	requests.c	requests.h	manager/serveur
Timer event management facility.	timeout_queue.c	Timeout_queue.h	manager/serveur
Time calculation functions used by the time out facility.	time_func.c	time_func.h	manager/serveur

Table 8 *Location of the files containing the SNMP Session Manager logic*

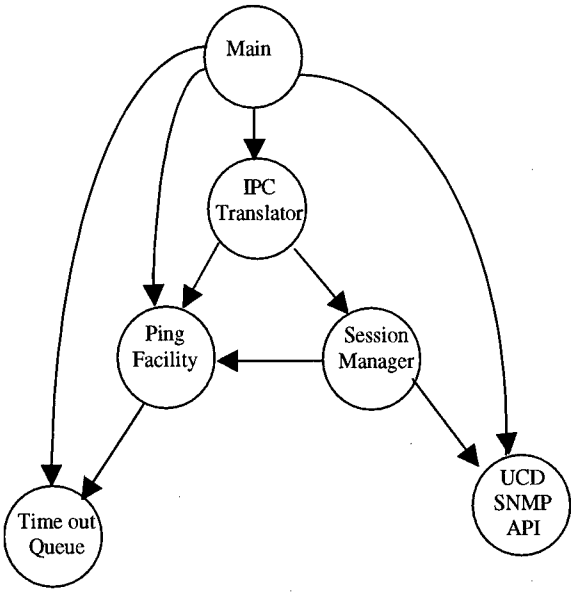


Figure 5 *SNMP Session Manager code dependencies*

5.3- Design and implementation of the Management Console's GUI

This subsection presents the design and implementation of the Management Console's GUI. The goal of this element is to provide the user access to the functionality of this network management solution through a Graphical User Interface, and to isolate all overhead incurred due to interface management, such as the redrawing of the windows.

As was already stated in the requirement analysis section, the GUI interface is written in Tcl/Tk so that it can be rapidly prototyped and modified independently of the underlying capabilities. However, it does need to handle receiving and sending messages through sockets, which are not sufficiently supported in Tcl/Tk to allow it to be used for this functionality. It is for this reason that a module implementing the management of the sockets in C, and a link between this C code and the Tcl/Tk interpreter, were written. As one of the goals of the design was to control when the socket is checked for received messages, the link module is the main loop of the GUI process. There is a specific set of steps defined in order to use Tcl/Tk in as the way that was just described and this case is no exception. The "init_ui" function, which is called during the initialisation phase, implements the required initialisation. The next necessary step is to regularly invoke the "Tcl_doOneEvent" function in order to process any Tcl/Tk events that may have been created by a user, or by the receipt of a message. In this case, through trial and error, it was determined that calling "Tcl_doOneEvent" once every 50 milliseconds was sufficient.

The remaining step needed for this integration is to implement a way to send the user's requests to the code controlling the communication socket, which is written in C, and to forward the results to the GUI display functions, which are written in Tcl/Tk. To allow the Tcl/Tk code to send a request through the socket to the session manager, a new Tcl/Tk command was added in the "init_ui" function. This new command is named "send_session_manager" and it is implemented in C in a function of the same name. In the opposite direction, the information contained in a message received on the socket is sent to the Tcl/Tk interpreter through the "Tcl_Eval" Tcl procedure. This procedure interprets the Tcl/Tk command contained in the string it receives. Since the messages received on the socket are strings, a simple Tcl/Tk procedure was written for this purpose (recmsg) and the message received from the socket, preceded with the string "recmsg", is given as the command parameter to "Tcl_Eval". All of the C code in this Tcl/Tk to C interface is in the module "UI_main.c".

The remaining code developed for the GUI is in Tcl/Tk. It is difficult to describe the code logic since so much behaviour is intrinsic to the object type definitions in Tcl,/Tk and to the order or hierarchy in

which the objects are declared. The remainder of this section will nevertheless try to describe the Tcl/Tk code. However, the description will be based on the structure of the interface rather than on the logic behind it. Table 9 summarises the different files implementing the GUI and their location in the code subdirectories.

Description	Source Code file	File location
Code that implements the link between C and Tcl and manages the communication sockets.	UI_main.c	manager/serveur
Definition of generic routines and of the main window where all of the panels are incorporated into a single interface.	UI_main.tcl	manager/serveur
Definition and implementation of the network map panel.	UI_network_map.tcl	manager/serveur
Definition and implementation of generic router panels.	UI_router.tcl	manager/serveur
Definition and implementation of ELOS panels.	UI_ELOS.tcl	manager/serveur
Graphical based ping tool in a panel.	UI_ping.tcl	manager/serveur
Utility module that allows the display of hint windows when the cursor remains immobile over a widget.	balloon.tcl	manager/serveur
Utility module for the display of bar graphs with an automatically adjusting scale inside a window.	bargraph.tcl	manager/serveur
Utility module that allows the display of paned resizable sub-windows.	panedwin.tcl	manager/serveur
Utility module that allows the display of large windows within a smaller display area, using scroll bars when needed.	scrollcanvas.tcl	manager/serveur
Utility module that defines a generic table display and modification panel.	tables.tcl	manager/serveur
Utility module that allows the display of several windows in the same space, where the one displayed is selected with the help of tabs.	tabwindows.tcl	manager/serveur

Table 9 *Location of the files used by the Management GUI logic*

The code has been structured to be as modular as possible, and several modules have been defined expanding stock widgets to simplify the definition of the core windows. The way to use these super objects is well documented in the comments in the beginning of the code, and there is no reason to dwell on their definition beyond the description in Table 9. These modules are "balloon.tcl", "panedwin.tcl", "tabwindows.tcl", "scrollcanvas.tcl", "bargraph.tcl", and "tables.tcl".

The main window is divided into three portions: a device selection area in the upper left corner, a device log area in the upper right corner, and a tool area in the lower half. This division is defined using the

super object created in "panedwin.tcl", which allows these sections to be resized by dragging a tab. The device selection area allows the manager to select the device to which any action will apply from a list of all available devices. The device log area is a list of events that have occurred on the currently selected device. The tool area is further divided into several sub-windows, each displaying a different category of tool. Each of these tool category sub-windows can be reached by selecting the appropriate tab. These tabs conform to the analogy of a filing cabinet folder, and are created and managed by the super object defined in "tabwindows.tcl'. The code creating the main window of the interface is in procedure "buildMainInterface"of the module "UI_main.tcl".

There are in all six tabs in the tools area, two of which are meant to operate on the whole network in general: network map and network tools. The other tabs are meant to operate on a single device and they include a device configuration page, a performance display page, an alarms configuration page, and a logs configuration page.

The network map page is defined and managed in "UI_network_map.tcl". Its function is to display the network architecture using icons, and to display the status of the network using colours that denote the status of each individual device and link. The user can also select the device upon which any action will apply from the network map just as in the device selection area. In its current incarnation, it is very limited and only displays a fixed number of hard coded devices. Future work could add to it auto-discovery functionality and a topology drawing tool.

At the moment, the network tools page includes only a ping tool. This tool is composed of a couple of simple input boxes which allow the user to set parameters, a start/stop button, and an area in which to display the results from the execution of the request sent to the SNMP session manager. It is defined and managed in "UI_ping.tcl".

The format of the pages used to manage a particular device is dependent on the type of that device. To simplify the management of the display of the devices and appropriate values, every managed device has its own set of Tcl/Tk objects, and they are simply unmapped or mapped as needed. When a new device is selected, be it by the network map or through the device selection area, all of the pages are changed to this device's instances using the procedure "switchdevice_by_selection_box", or "switchdevice", both of which are defined in the module "UI_main.tcl". The pages changed include the device log area, and all four device dependent tabs in the tool area. These pages are created when a device is added to the list of manageable devices by the procedure "adddevice" defined in the module "UI_main.tcl". This procedure calls a set of functions that create the appropriate pages. Every type of device must define this set of functions to be compatible with the defined interface scheme. There are currently two types of devices that have some of these functions defined; these are the ELOS SNAC in module "UI_ELOS.tcl", and a generic router in module "UI_router.tcl".

Both device type implementation modules have a similar code structure. Each of them includes sets of constructs that fall in the following groups:

- A set of functions that individually define a UI panel;
- A set of function that implement the actions taken when a button in the UI panels is pressed;
- A set of functions for each message that can be received from the SNMP Session Manager, such as "get_answer", "get_failure", or "set_answer";
- A group of constant declarations describing the MIB variables used.

The first set of functions are fairly regular Tcl constructs in which there are buttons, scroll bars, and editable fields used to display and change values. All of the panels follow a template where the commands area is located on the left, and the variable display is located on the right. The editable fields are local copies, or interpretations, of the managed device's MIB variables. When a button in the command area is pressed the function implementing the appropriate action is invoked, and an SNMP message is sent to the managed device's Agent with the help of the SNMP Session Manager. When a response is received from the managed device's Agent, the corresponding function is invoked, and it interprets and displays the

received data on the appropriate panel. Finally, the MIB declarations are used for the translation of variable names in order to send and interpret the SNMP data.

"Annex C: Screen captures of GUI" presents a screen capture of each window in this GUI.

5.4- Design of the Log Manager

This subsection presents the responsibilities intended for the SNMP Log Manager as a reference for future work. Its implementation should follow these processing steps:

- Open sockets for communication with the SNMP Session Manager application;
- Read its configuration data files in order to initiate periodic requests to the SNMP Session Manager;
- Open the log files;
- Open a socket to receive logs that must be saved to disk;
- Wait for requests and serve them while making periodic log requests (as defined in the configuration files) to the SNMP Session Manager. These requests can be for:
  - Logs that must be appended to the files;
  - Access for data in the files;

5.5- Design of the SNMP Trap Monitor

This section presents the responsibilities intended for the SNMP Trap Monitor. Its implementation should be a simple enough task as it consists of these three basic processing steps:

- Open sockets for communication with the other applications;
- Open the SNMP Trap port (number 162) with the help of the NET-SNMP interface;
- Wait for Trap messages on the open port and forward the received Trap information to the other processes.

6- Future Work

As the development effort progressed, it became clear that further development could not proceed by simply extending the current implementation. In other words, development has reached a crossroads, in the sense that several key issues need to be reviewed, and the best route to take for future efforts must be determined. For instance, the current platform for the SNAC cards (VME card cages) has become obsolete and very expensive to support. A new version on another platform is therefore required before any follow up work can be performed. Another issue is the fact that the field of network management has evolved and several tools that could be incorporated into a network management solution have become available in the public domain. A carefully selected group of these tools incorporated in future designs could greatly reduce the costs of ownership. This section presents some thoughts on the direction of future study and development.

The issue that is most pressing for any future work is the development of an automatic device discovery algorithm appropriate for use over low bandwidth links. This issue must be addressed as it is completely ignored by the providers of commercial network management solution. Such an algorithm would need to have some inherent knowledge of the composition of the network being monitored. Three examples provide promising directions for future development. The first of these uses the fact that the network being managed is a network where only the topology between sub-networks (individual navy ships) changes, not the sub-networks themselves. Due to this fact, a set of static sub-network description files could be used to "discover" the devices on the network as ships come within range, or units deploy and add or remove themselves from the network. The second possible direction for development uses the fact that OSPF is used between the sub-networks. This means that the Link State Advertisements (LSA) table of the OSPF MIB on the local edge router could be used along with SNMP Traps to detect topology changes. Finally, the sub-network edges provide promising insertion points for a passive TCP/IP Finger Printing algorithm, which could be used to discover topology or device status changes. It is important to note that potential solutions to the automatic device discovery algorithm are also applicable to the discovery of a remote device's status.

The problem of managing a network from a remote location also needs attention as it has specific problems in military networks containing low bandwidth links and a rapidly evolving network topology. Remote network management solutions being considered try to reduce the amount of traffic on the low bandwidth links. These include having an operational data repository that keeps only recent data in order to reduce the amount of logs that must be transferred for any initial assessment of the causes for the current state of the network. One could also create super requests that translate into several SNMP commands when executed on a local LAN, or decide to use multicast communications for certain requests. Finally, handling alarms on the local network in order to filter them before they are sent on to the remote management site is another example of such a solution. All of these solutions may well be acceptable and desirable in the long run. However, they require fine adjustments of their parameters such as utilisation frequency, time intervals, and scope. For example, what time interval between subsequent writings of performance logs provides enough information to properly diagnose a problem? Which super requests make sense? Could an existing alarm filtering solution be adapted? It is important to note that a solution could be useful on one type of SNAC and detrimental on another. As a result of all of these factors, rapid prototyping and data collection will be required to provide a better gauge of the processing power, transmission delay, and bandwidth cost or benefit of different solutions.

In conclusion, given that the network management tools available in the open source community are changing, future development should concentrate on resolving the issues specific to the network at hand, and in obtaining results that can be reengineered rapidly. The remaining issues that need to be addressed to provide a basic solution are:

- Handling of Traps on the Network Management Console;
- Auto-discovery of devices in low bandwidth environments with changing topologies;
- Automatic generation of the network map;

- The ability to request statistics and manage logs from the Network Management Console;
- Security mechanisms (ranging from migrating to SNMP-V3 to user authentication controls on the console);
- Development of a configuration tool for the Network Management Console.

Several additional items should also be considered. These are:

- A redesign of the SNAC code base;
- Porting the Agent hooks to the new SNAC code base;
- Changing the Agent engine to a public domain implementation of SNMP-V3 such as Net-SNMP [4].

7- Conclusion

This final report provides a record of the network management effort of the "Sub-Network Access Control Technology Demonstrator" project. The development efforts described in this document have produced a prototype implementation of a SNAC Management Console as well as Agents for the ELOS and BLOS SNAC cards. This constitutes a significant step towards the development of a complete network management system for a naval force network containing low bandwidth radio links. Additionally, this research and development activity has provided the opportunity to identify and explore general issues for network management in low bandwidth environments with changing topologies.

Of all of the identified issues, the one that is the most important is the bandwidth used by the network management application, as it is from this issue that most of the others originated. Today's automatic device discovery algorithms, as well as potential remote management applications, are two functions that may cause difficulties in the typically low bandwidth military networks. It is important to note that the scope of the design given in this document applies only to a ship node, and therefore only addresses local management. Network management from a remote location connected to the network via low bandwidth links presents several additional problems that have only been partially examined.

8- References

[1] RFC 1155: "Structure and identification of management information for TCP/IP-based internets"

[2] RFC 2570: Introduction to Version 3 of the internet-standard Network Management Framework

[3] RFC 1213: Management information base for Network Management of TCP/IP based Internets: MIB-II

[4] Net-SNMP project; please refer the open source project's WWW page at address: http://net-snmp.sourceforge.net/

Annex A Acronyms

AS	Autonomous System
BLOS	Beyond Line Of Sight radio link
C4	Command, Control, Communications and Computers
CSNI	Communications Systems Network Interoperability
CWAN	Coalition Wide Area Network
ELOS	Extended Line Of Sight radio link
EMCON	EMission CONtrol
FCAPS	Fault, Configuration, Accounting, Performance and Security management
FTP	File Transfer Protocol
GUI	Graphical User Interface
HF	High Frequency
LAN	Local Area Network
MIB	Management Information Base
IP	Internet Protocol
JWID	Joint Warrior Inter-operability Demonstration
RF	Radio Frequency
SHF	Super-High Frequency
SNAC	Sub-Network Access Controller
SNMP	Simple Network Management Protocol
SNMP-V3	Simple Network Management Protocol Version 3
SRIU	Sub-Router Interface Unit
TD	Technology Demonstrator
TGAN	Task Group Area Network
UDP	User Data Protocol
UHF SATCOM	Ultra High Frequency SATellite COMmunication
VME	Versa Module Europa (as defined in the IEEE 1014-1987 standard)

Annex B Supplemental MIB Variables definition file

```
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- *
-- * CRIU AGENT MIB
-- * Version 5/04/00
-- *
-- * Modified 6/15/96
-- * Modified 8/13/96 by kvo to change TRAP-TYPE to NOTIFICATION-TYPE
-- * Add Priority, application for appStat and changed range to some integer
-- * values 9/16/96 TT.
-- * Modified 06/13/00 by [CRC/mjb] to reflect MISN and CA additions.
-- * Modified 03/01/01 by [CRC/HLJ] to update MISN changes and to remove indexing
-- * by cap Ip address of capAdmin, capStats and appStats.
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


CRIU-MIB DEFINITIONS ::= BEGIN

IMPORTS
        OBJECT-TYPE, MODULE-IDENTITY,
        NOTIFICATION-TYPE
                FROM SNMPv2-SMI;



navy                    OBJECT IDENTIFIER ::=    { enterprises 1738 }
navyADNS                 OBJECT IDENTIFIER ::=    { navy 2 }
criuMib                 OBJECT IDENTIFIER ::=    { navyADNS 300 }
criuAdmin               OBJECT IDENTIFIER ::=    { criuMib 1 }
capAdmin                OBJECT IDENTIFIER ::=    { criuMib 2 }
capStats                OBJECT IDENTIFIER ::=    { criuMib 3 }
appStats                OBJECT IDENTIFIER ::=    { criuMib 4 }
criuTraps               OBJECT IDENTIFIER ::=    { criuMib 5 }
priority                OBJECT IDENTIFIER ::=    { criuMib 6 }


-- Module Identification Definition.

criuMIB         MODULE-IDENTITY
                LAST-UPDATED "0006130000Z"
                ORGANIZATION "CRC"
                CONTACT-INFO
                        "       Henryk Lukasik

                         Postal: CRC Canada
                                 PO BOX 11490 Stn H
                                3701 Carling Ave
                                 Ottawa Ontario  K2H 8S2
                                 Canada

                            Tel: +1 613 998 5240
                            Fax: +1 613 998 9648

                         E-mail: henryk.lukasik@crc.ca"
                DESCRIPTION
                        "This is the CRIU MIB module.  It is provided as
                         an example of how to make extensions to the VxWorks SNMP
                        Agent MIB.  This module contains the modification made to
                        accomodate the Canadian MISN requirements."

                ::= { criuMib 7 }


-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
-- *
-- *   CRIU Administration Group        (criuAdmin)
-- *
-- *  (enterprises.src.criuMib.1)
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

criuIpAddress           OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " IP Address of CRIU SNMP agent "
                        ::= { criuAdmin 1 }

criuHostName            OBJECT-TYPE
                        SYNTAX DisplayString (SIZE(0..255))
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Host name of CRIU "
                        ::= { criuAdmin 2 }


logState                OBJECT-TYPE
                        SYNTAX  INTEGER { on (1), off (2) }
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Determines if information will be logged within the CRIU.
"
                        ::= { criuAdmin 3 }


logDuration             OBJECT-TYPE
                        SYNTAX INTEGER (1..5000)
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Range of log duration.  Unit is # of Log Msg.  "
                        ::= { criuAdmin 4 }


fileName                OBJECT-TYPE
                        SYNTAX OCTET STRING (SIZE(0..255))
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " File name that CRIU will be logged."
                        ::= { criuAdmin 5 }

criuReboot              OBJECT-TYPE
                        SYNTAX  INTEGER { reboot (1), normal (2) }
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Reboot CRIU when called "
                        ::= { criuAdmin 6 }



-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- *
-- *   CAP Administration Group (capAdmin)
-- *
-- *  (enterprises.src.criuMib.2)
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

------------------------------------------------------------
--      CAP Administration
```

```
-------------------------------------------------------------

capHostName             OBJECT-TYPE
                        SYNTAX DisplayString (SIZE(0..255))
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " CAP's name Example: SHF, UHF etc."
                        ::= { capAdmin 1 }

capLinkDataRate                 OBJECT-TYPE
                        SYNTAX INTEGER (1..10000000)
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                " Displays speed of the links in kBytes "
                        ::= { capAdmin 2 }

capId                   OBJECT-TYPE
                        SYNTAX OCTET STRING (SIZE(4))
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Assigns a unique identifier to CAP.  "
                        ::= { capAdmin 3 }

queueReport             OBJECT-TYPE
                        SYNTAX INTEGER (1..200000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " How often CAP sends Queue Report to CRIU. Unit is in
bytes.  "
                        ::= { capAdmin 4 }

queueSize               OBJECT-TYPE
                        SYNTAX INTEGER (1..200000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Identifies the CAP queue size in bytes."
                        ::= { capAdmin 5 }

queueThreshold          OBJECT-TYPE
                        SYNTAX INTEGER (1000..200000)
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Sets the queue threshold in bytes for the CAP."
                        ::= { capAdmin 6 }


icmpThreshold           OBJECT-TYPE
                        SYNTAX INTEGER (1..200000)
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Sets the ICMP threshold in bytes."
                        ::= { capAdmin 7 }

udpThreshold            OBJECT-TYPE
                        SYNTAX INTEGER (1..200000)
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Sets the UDP threshold in bytes."
                        ::= { capAdmin 8 }

tcpRejDupTime           OBJECT-TYPE
                        SYNTAX INTEGER (1..200000)
                        MAX-ACCESS  read-write
```

```
                        STATUS  current
                        DESCRIPTION
                                " Sets the time threshold to begin dropping
                                    duplicate TCP packets. Unit is secconds."
                        ::= { capAdmin 9 }

capStatisticReset       OBJECT-TYPE
                        SYNTAX INTEGER { reset (1) }
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Reset CAP statistic."
                        ::= { capAdmin 10 }

capStatus                       OBJECT-TYPE
                        SYNTAX INTEGER { capUp(2),capDown(1),capEmcon(7) }
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " CAP Operational status."
                        ::= { capAdmin 11 }

emconMode                       OBJECT-TYPE
                        SYNTAX INTEGER { on (1),off (2) }
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Put the CAP into EMCON mode if ON.  If OFF,
                                    the CAP will be back to normal mode."
                        ::= { capAdmin 12 }

emconMethod             OBJECT-TYPE
                        SYNTAX INTEGER { bypassRouter (1),modifySrcIP (2) }
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                                " Set the method which the CRIU use to receive
                                    data when the CAPs are in EMCON."
                        ::= { capAdmin 13 }


-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- *
-- *   CAP Statistics Group     (capStats)
-- *
-- *  (enterprises.src.criuMib.3)
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

-------------------------------------------------------------
--      CAP Statistics
-------------------------------------------------------------


dataBytesTransmitted        OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Number of data bytes transmitted by the CAP
                                   within last 60 secs. "
                            ::= { capStats 1 }

dataBytesAcked              OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Number of data bytes acked by the CAP within
                                   last 60 secs. "
                            ::= { capStats 2 }
```

```
dataBytesDropped            OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Number of data bytes dropped by the CAP within
                                  last 60 secs. "
                            ::= { capStats 3 }

dataBytesUnacked            OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Number of data bytes unacked by the CAP within
                                  last 60 secs. "
                            ::= { capStats 4 }

dataBytesReceived           OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Number of data bytes received by the CAP within
                                  last 60 secs. "
                            ::= { capStats 5 }

dataBytesQueued                  OBJECT-TYPE
                           SYNTAX INTEGER (0..10000000)
                           MAX-ACCESS  read-only
                           STATUS  current
                           DESCRIPTION
                                 " Total number of data bytes queued by the CAP within
                                   last 60 secs. "
                           ::= { capStats 6 }


linkThroughput              OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Average link throughput of the CAP within last
                                  60 secs. "
                            ::= { capStats 7 }

linkUtilization                    OBJECT-TYPE
                           SYNTAX INTEGER (0..100)
                           MAX-ACCESS  read-only
                           STATUS  current
                           DESCRIPTION
                              " Average link utilization by the CAP within last
                                 60 secs. "
                           ::= { capStats 8 }

totalDataBytesTransmitted    OBJECT-TYPE
                           SYNTAX INTEGER (0..10000000)
                           MAX-ACCESS  read-only
                           STATUS  current
                           DESCRIPTION
                              " Total number of data bytes transmitted by the CAP
                                since running or last reset. "
                           ::= { capStats 9 }

totalDataBytesAcked        OBJECT-TYPE
                           SYNTAX INTEGER (0..10000000)
                           MAX-ACCESS  read-only
                           STATUS  current
                           DESCRIPTION
                              " Total number of data bytes acked by the CAP since
```

```
                                running or last reset. "
                            ::= { capStats 10 }

totalDataBytesDropped       OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Total number of data bytes dropped by the CAP since
                                  running or last reset. "
                            ::= { capStats 11 }

totalDataBytesUnacked   OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Total number of data bytes unacked by the CAP since
                                  running or last reset. "
                        ::= { capStats 12 }

totalDataBytesReceived      OBJECT-TYPE
                            SYNTAX INTEGER (0..10000000)
                            MAX-ACCESS  read-only
                            STATUS  current
                            DESCRIPTION
                               " Total number of data bytes received by the CAP
                                  since running or last reset. "
                            ::= { capStats 13 }

totalDataBytesQueued          OBJECT-TYPE
                              SYNTAX INTEGER (0..10000000)
                              MAX-ACCESS  read-only
                              STATUS  current
                              DESCRIPTION
                                 " Total number of data bytes queued by the CAP
                                   since running or last reset. "
                              ::= { capStats 14 }


ospfBytesTransmitted    OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                               " Total number of OSPF bytes transmitted by the CAP."
                        ::= { capStats 15 }

ospfBytesDropped        OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                               " Total number of OSPF bytes dropped by the CAP."
                        ::= { capStats 16 }


ospfBytesReceived       OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                               " Total number of OSPF bytes received by the CAP."
                        ::= { capStats 17 }

sourceQuenchSent        OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                               " Total number of source quench  PDUs sent back
```

```
                        to applications."
                    ::= { capStats 18 }

bytesLoadShared       OBJECT-TYPE
                    SYNTAX INTEGER (0..10000000)
                    MAX-ACCESS  read-only
                    STATUS  current
                    DESCRIPTION
                           " Total number of bytes load shared."
                    ::= { capStats 19 }


-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- *
-- *   Application Statistics Group     (appStats)
-- *
-- *  (enterprises.src.criuMib.4)
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

-------------------------------------------------------------
--      Application Statistics Table
-------------------------------------------------------------

appStatsTable         OBJECT-TYPE
                      SYNTAX    SEQUENCE OF AppStatsEntry
                      MAX-ACCESS    not-accessible
                      STATUS    current
                      DESCRIPTION
                        "Table object corresponding application statistics entries. "
                      ::=  { appStats 1 }

appStatsEntry         OBJECT-TYPE
                      SYNTAX    AppStatsEntry
                      MAX-ACCESS    not-accessible
                      STATUS    current
                      DESCRIPTION
                        "Table entry containing Application Statistical information "
                      INDEX   { applicationIndex}
                      ::=  { appStatsTable 1 }


AppStatsEntry    ::=  SEQUENCE {
                 applicationIndex              INTEGER (0..4),
                portNumber                    INTEGER (0..65535),
                sourceIP                      IpAddress,
                appsDataBytesTransmitted          INTEGER (0..10000000),
                appsDataBytesReceived             INTEGER (0..10000000),
                appsDataBytesDropped              INTEGER (0..10000000),
                applicationName                    OCTET STRING
                }


applicationIndex      OBJECT-TYPE
                      SYNTAX  INTEGER (0..4)
                      MAX-ACCESS  read-only
                      STATUS  current
                      DESCRIPTION
                        " Used to index list of applications to identify
                          associated statistics."
                      ::= { appStatsEntry 1 }

portNumber            OBJECT-TYPE
                    SYNTAX INTEGER (0..65535)
                    MAX-ACCESS  read-only
                    STATUS  current
                    DESCRIPTION
                      " Port number of application. "
```

```
                        ::= { appStatsEntry 2 }

sourceIP                 OBJECT-TYPE
                        SYNTAX IpAddress
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                          " IP address of application.  Note that either port number
                          or source IP will be set to 0."
                        ::= { appStatsEntry 3 }

appsDataBytesTransmitted    OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Transmitted data bytes associated with application."
                        ::= { appStatsEntry 4 }

appsDataBytesReceived       OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Received data bytes associated with application."
                        ::= { appStatsEntry 5 }

appsDataBytesDropped        OBJECT-TYPE
                        SYNTAX INTEGER (0..10000000)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                                " Dropped data bytes associated with application."
                        ::= { appStatsEntry 6 }
applicationName          OBJECT-TYPE
                         SYNTAX OCTET STRING (SIZE(0..255))
                         MAX-ACCESS  read-write
                         STATUS  current
                         DESCRIPTION
                           " Name of application."
                         ::= { appStatsEntry 7 }


-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- *   Priority Assignment Group     (priority)
-- *
-- *  (enterprises.src.criuMib.6)
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-------------------------------------------------------------
--      Priority Assignment Table
-------------------------------------------------------------

priorityTable           OBJECT-TYPE
                        SYNTAX    SEQUENCE OF PriorityEntry
                        MAX-ACCESS  not-accessible
                        STATUS    current
                        DESCRIPTION
                          "The table object corresponding to priority assignment
                           entries. "
                        ::=  { priority 1 }

priorityEntry           OBJECT-TYPE
                        SYNTAX    PriorityEntry
                        MAX-ACCESS  not-accessible
                        STATUS    current
                        DESCRIPTION
                          " A table entry containing priority assignment
                            information "
                        INDEX   { priorityIndex }
                        ::=  { priorityTable 1 }
```

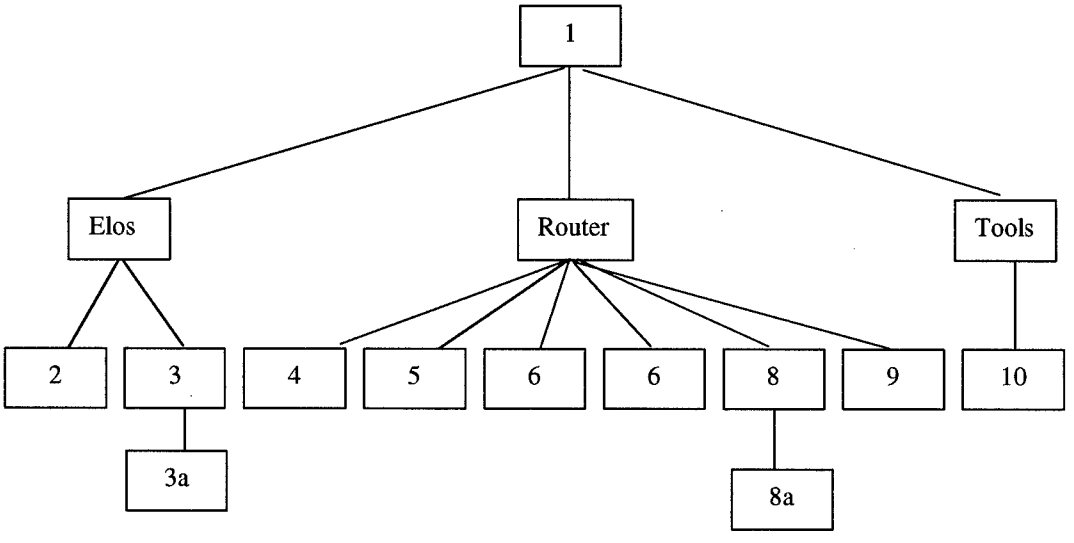
```
PriorityEntry    ::=  SEQUENCE {
                priorityIndex                   INTEGER,
                priPortNumber                      INTEGER,
                priSourceIP                        IpAddress,
                priapplicationName                 OCTET STRING,
                priorityLevel                   INTEGER

                }

priorityIndex           OBJECT-TYPE
                        SYNTAX  INTEGER (0..39)
                        MAX-ACCESS  read-only
                        STATUS  current
                        DESCRIPTION
                          " Used to index list of applications that will be
                            assigned a priority. "
                        ::= { priorityEntry 1 }

priPortNumber              OBJECT-TYPE
                        SYNTAX INTEGER (0..65535)
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                          " Port number of application. "
                        ::= { priorityEntry 2 }

priSourceIP                OBJECT-TYPE
                        SYNTAX IpAddress
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                          " IP address of application or host.  Note that either port
                            number or source IP will be set to 0."
                        ::= { priorityEntry 3 }

priapplicationName         OBJECT-TYPE
                        SYNTAX OCTET STRING (SIZE(0..255))
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                          " Name of application."
                        ::= { priorityEntry 4 }


priorityLevel           OBJECT-TYPE
                        SYNTAX INTEGER (0..15)
                        MAX-ACCESS  read-write
                        STATUS  current
                        DESCRIPTION
                          " Priority level assigned to application or host."
                        ::= { priorityEntry 5 }


-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- *
-- *   CRIU MIB Notifications
-- *
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


snacUp                  NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                          STATUS current
                          DESCRIPTION
                             "Trap indicating device has finished booting and
                               is now ready."
                          ::= { criuTraps 1 }

snacDown                NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
```

```
                        STATUS current
                        DESCRIPTION
                           "Trap indicating device is going off-line."
                        ::= { criuTraps 2 }

snacLinkUp              NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                            "Trap indicating connection with remote SNAC
                              established."
                        ::=  { criuTraps 3 }

snacLinkDown            NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating connection with remote SNAC lost."
                        ::=  {criuTraps 4 }

snacEMCONOn             NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating EMCON mode activated on SNAC."
                        ::=   {criuTraps 5 }

snacEMCONOff            NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating EMCON mode deactivated on SNAC."
                        ::=   {criuTraps 6 }


snacMessageQueueFull    NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating Message queue is full and
                              messages are being dropped."
                        ::= { criuTraps 7 }

snacMessageQueueNotFull       NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating message queue is no longer
                              full and messages are now being accepted. "
                        ::= { criuTraps 8 }

snacQueueAboveThreshold       NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating number of bytes in queue
                             has exceeded the threshold."
                        ::= { criuTraps 9 }


snacQueueBelowThreshold NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating number of bytes in queue
                              has dropped back below threshold."
                        ::= { criuTraps 10 }

snacLinkQualityPoor     NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
```

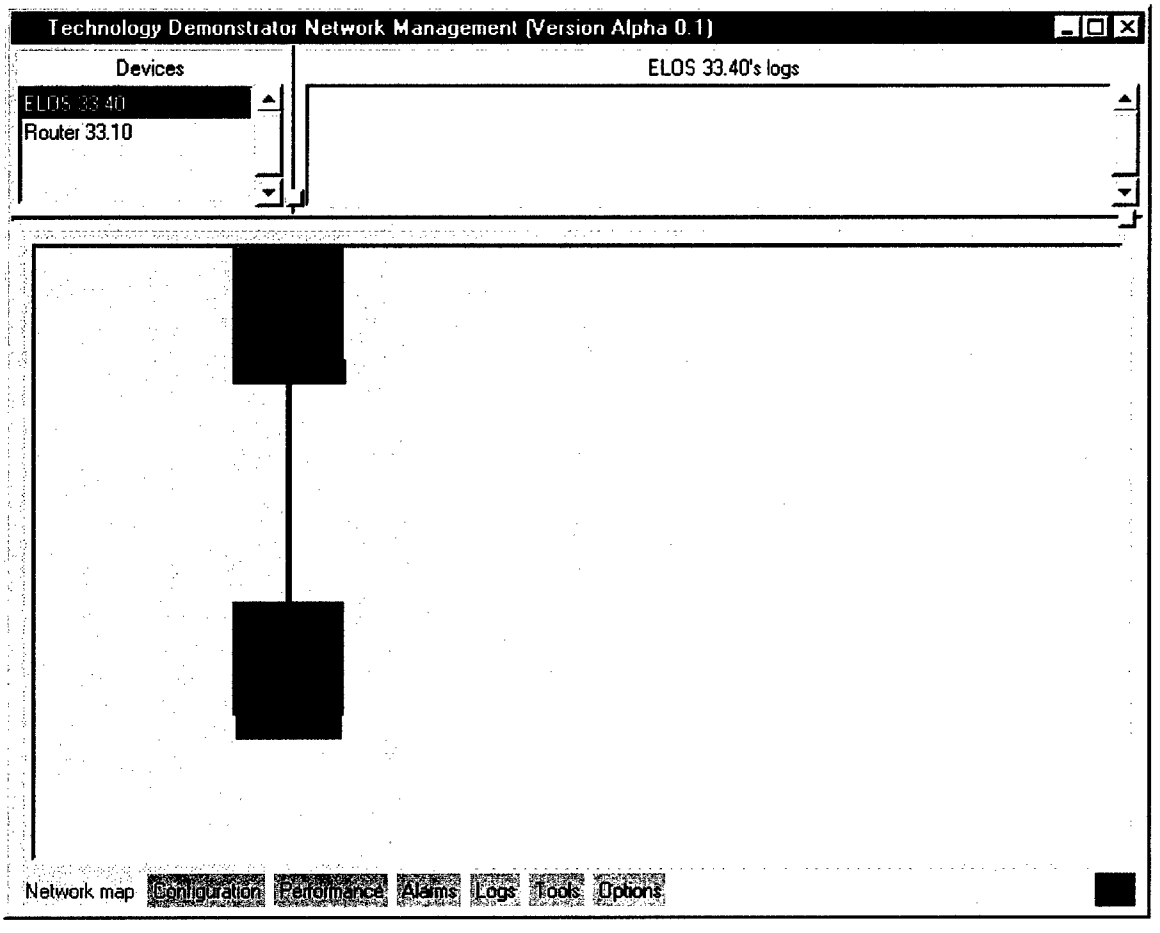
```
                        STATUS current
                        DESCRIPTION
                           "Trap indicating that the number of retransmitted
                             bytes has exceeded the threshold in the
                             last statistics period (i.e. 1 sec.). "
                        ::=  { criuTraps 11 }

snacLinkQualityGood     NOTIFICATION-TYPE
                        OBJECTS { criuIpAddress }
                        STATUS current
                        DESCRIPTION
                           "Trap indicating that the number of retransmitted
                             bytes has dropped back below the threshold
                             in the last statistics period (i.e. 1 sec.). "
                        ::=   { criuTraps 12 }

END
```

Annex C Screen captures of GUI

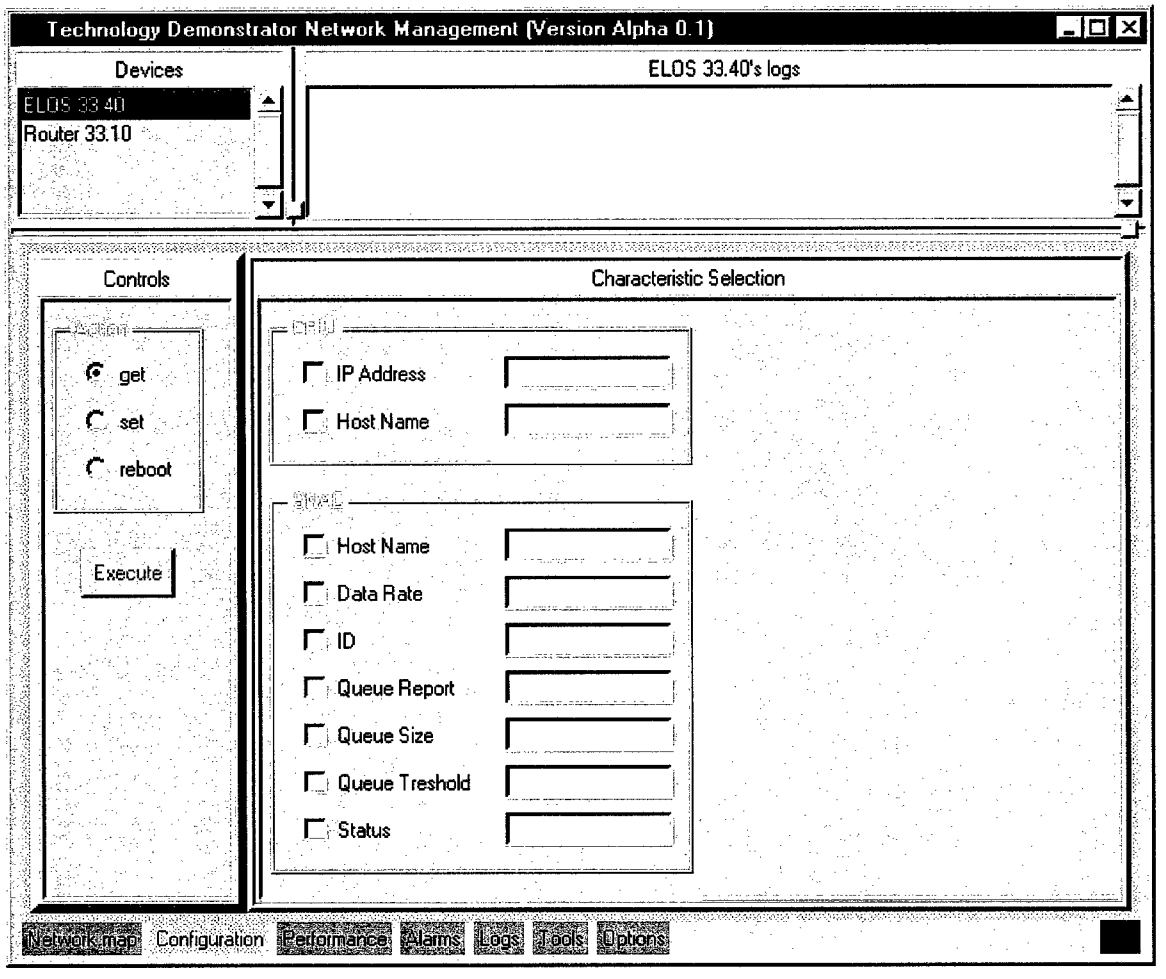
Screen classification of interface:



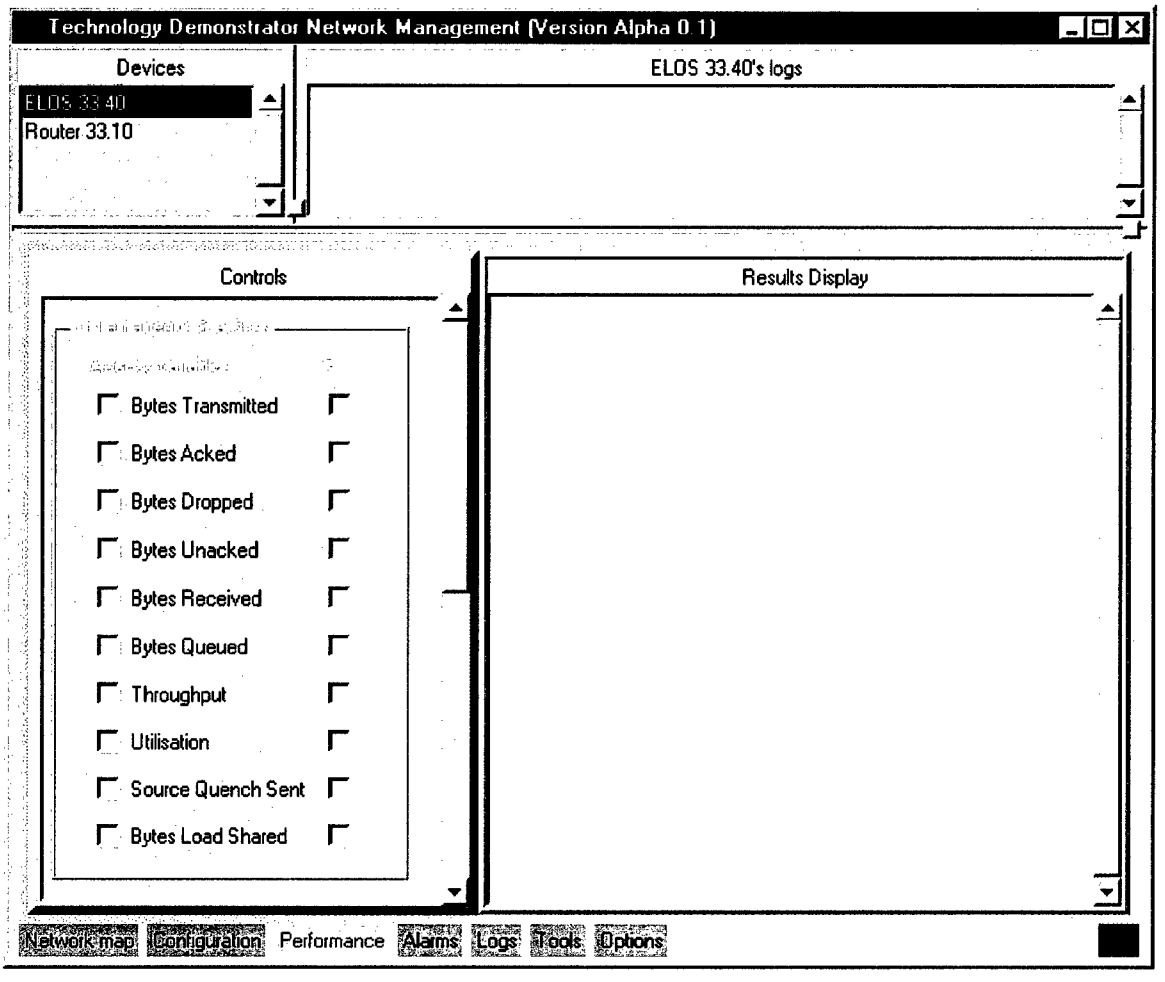
1: Network map
2: ELOS Configuration
3: ELOS instantaneous statistics.
3a: ELOS Cumulative statistics with graphing window
4: Router Interfaces table
5: Router IP Address table
6: Router IP Routing table
7: Router OSPF Configuration
8: Router OSPF Interface table
8a: Router OSPF Interface table(part 2)
9: Router OSPF Neighbors table
10: Ping tool



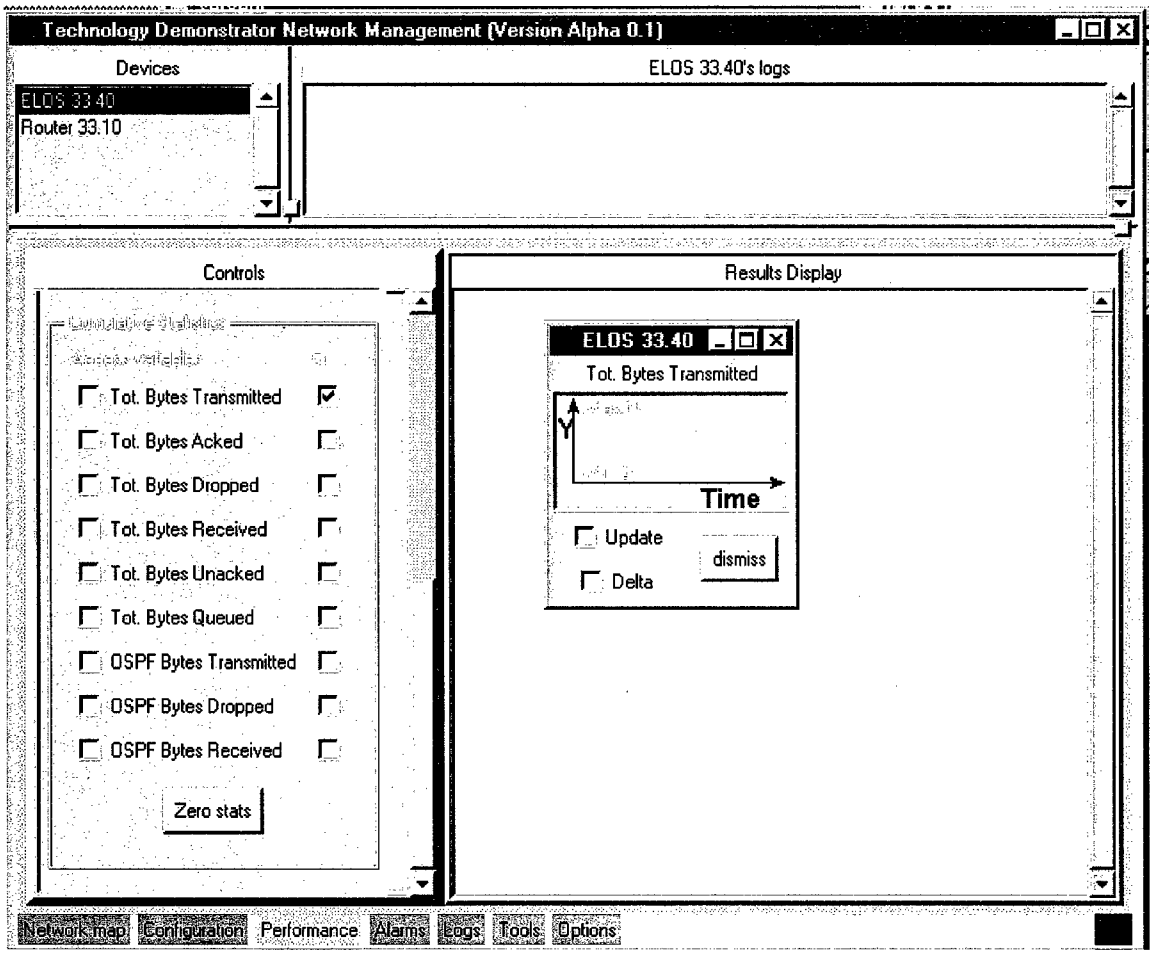
1. Network Map



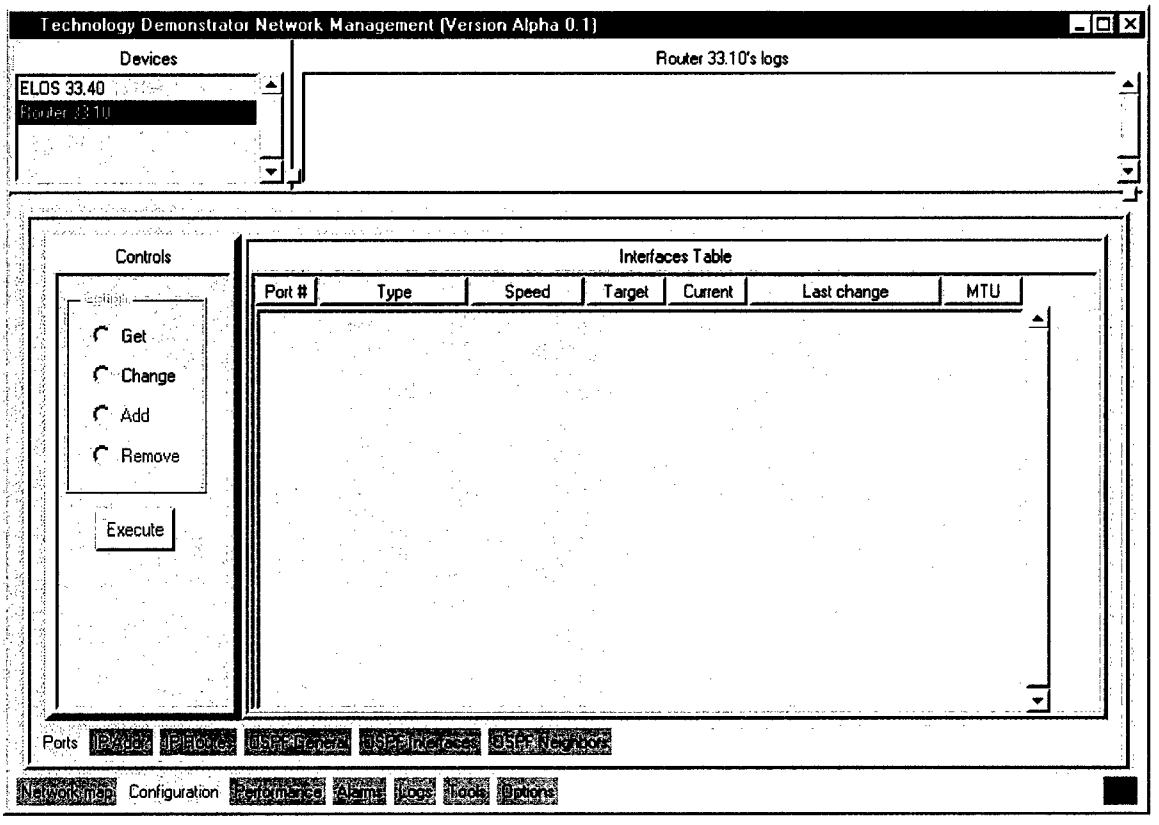
2. ELOS Configuration



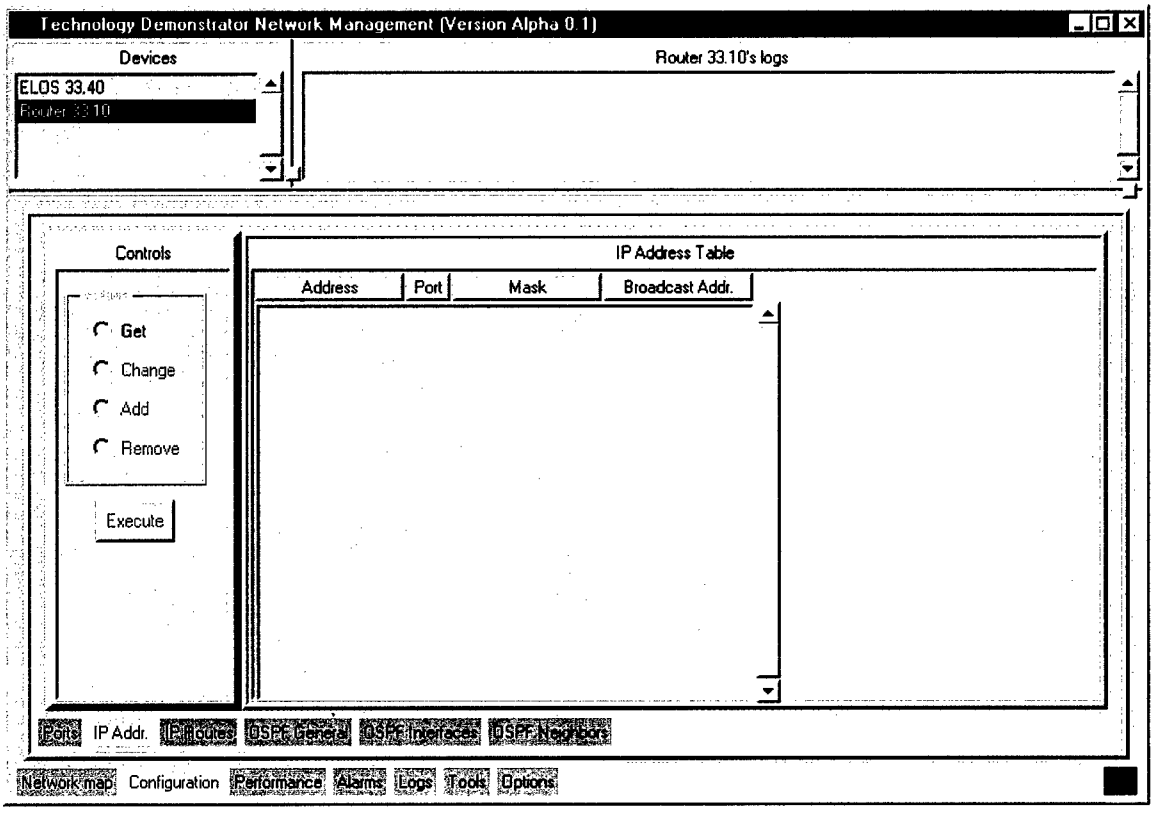
3: ELOS instantaneous statistics.



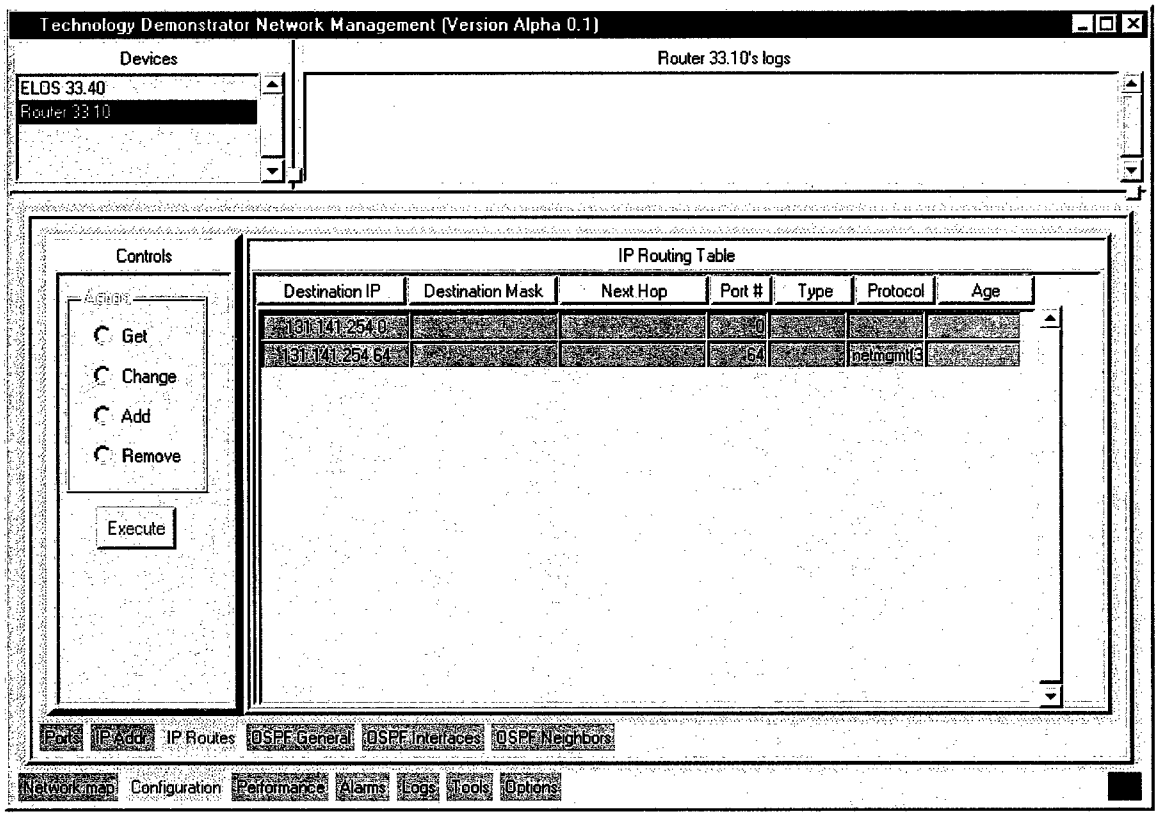
3a: ELOS Cumulative statistics with graphing window



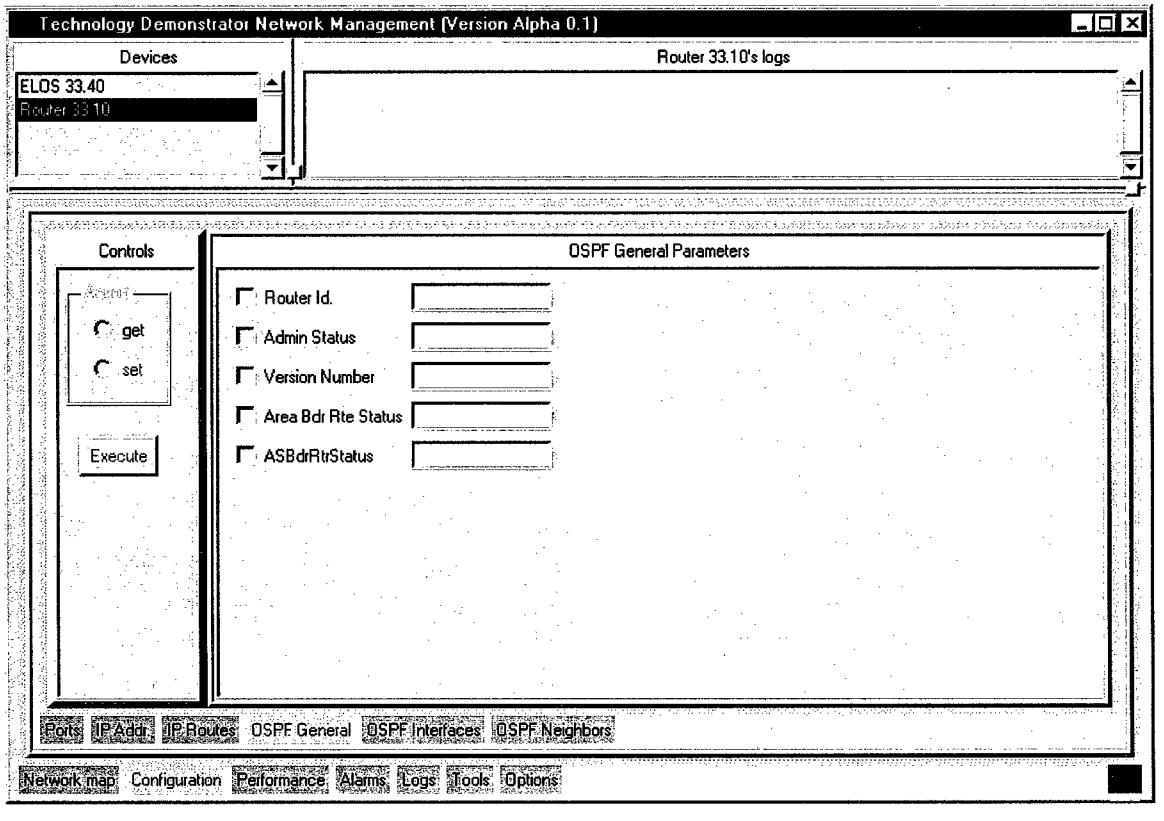
4: Router Interfaces table



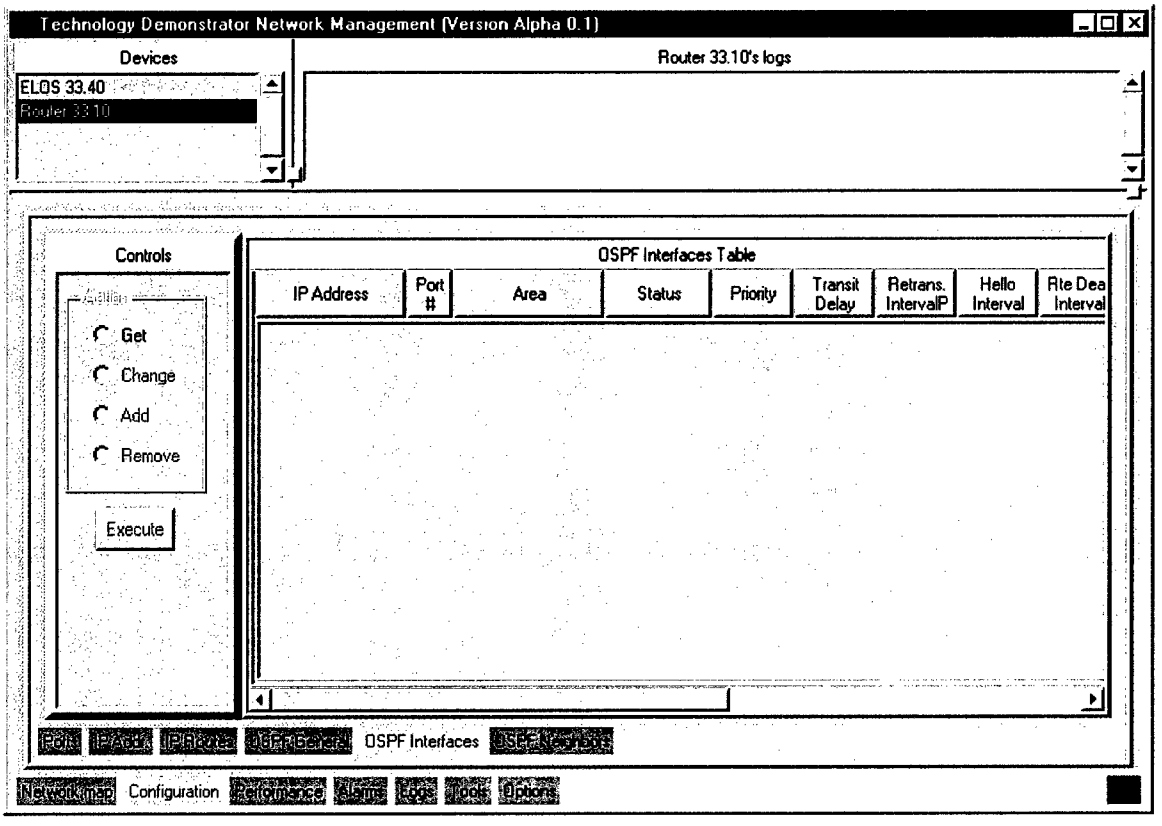
5: Router IP Address table



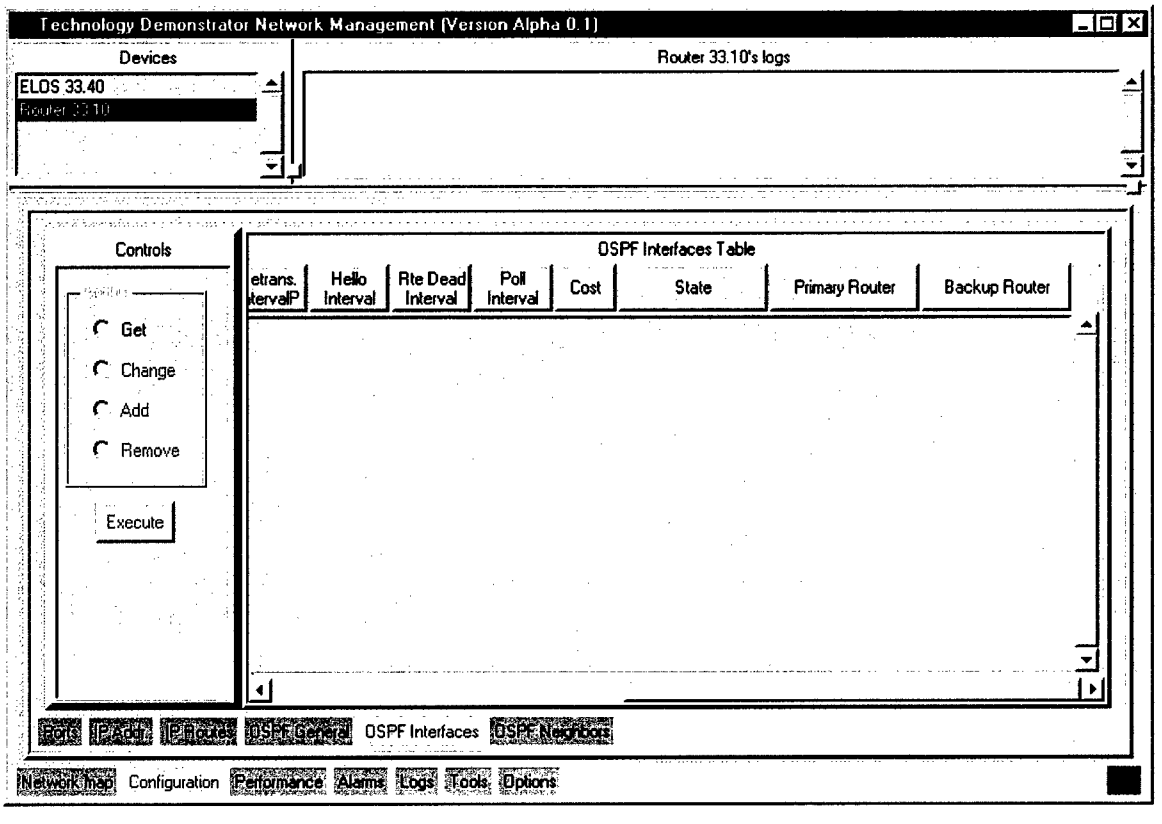
6: Router IP Routing table



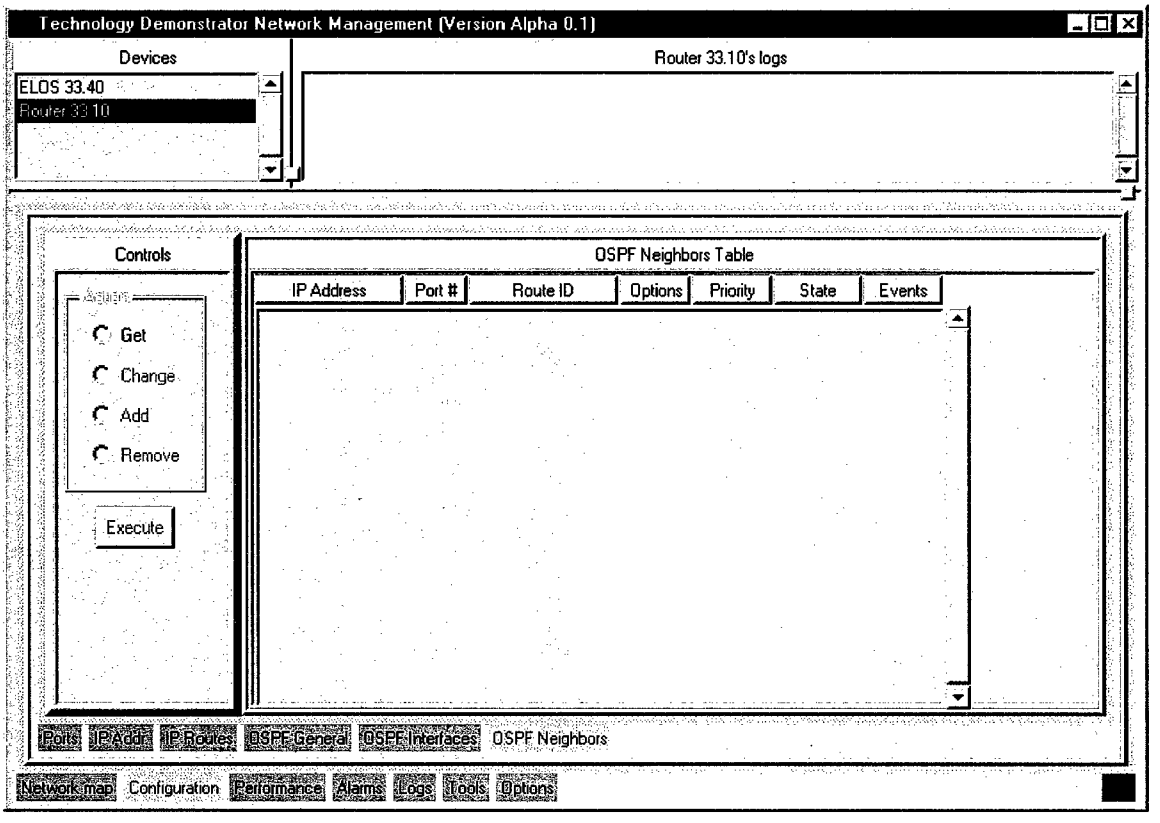
7: Router OSPF Configuration



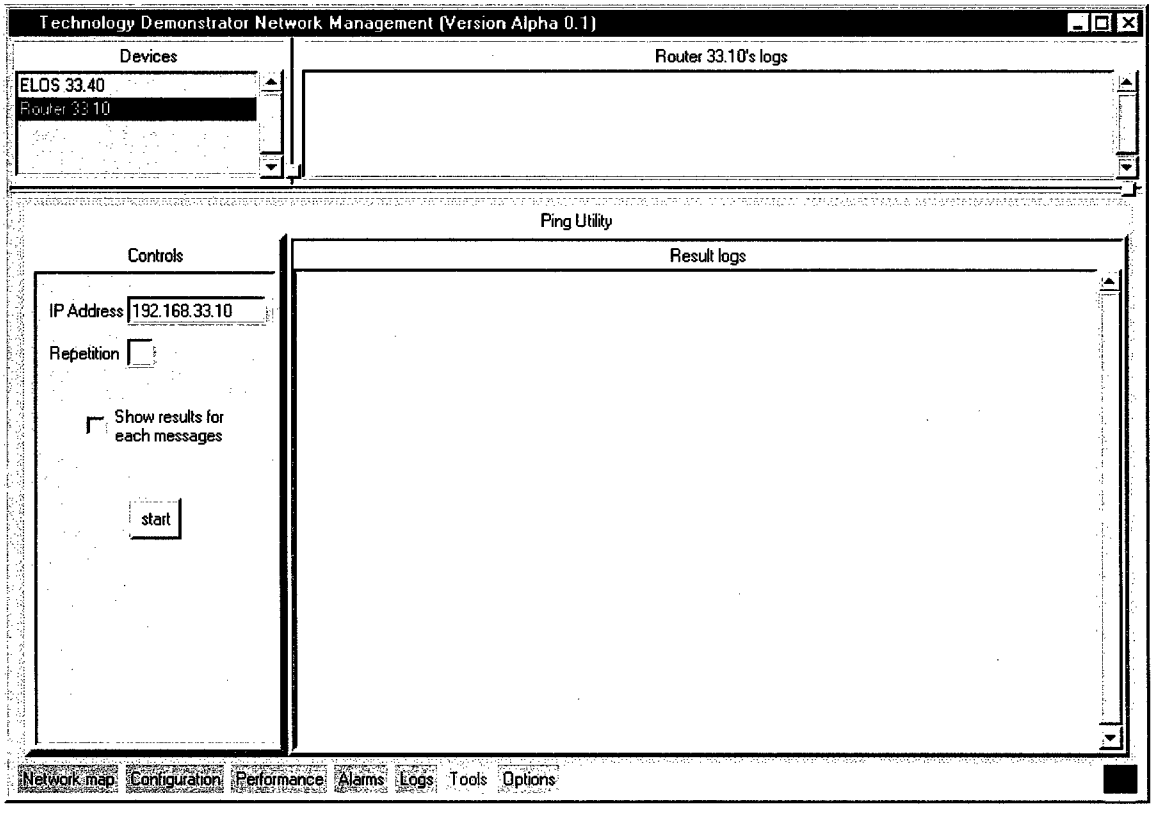
8: Router OSPF Interface table



8a: Router OSPF Interface table (part 2)



9: Router OSPF Neighbours table



10: Ping tool

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Communications Research Center
3701 Carling Ave., Box 11490, station H
Ottawa, Ontario K2H 8S2

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Sub-Network Access control Technology Demonstrator: Software Design of the Network Management system (U)

4. AUTHORS (Last name, first name, middle initial)

Lukasik, Henryk

5. DATE OF PUBLICATION (month and year of publication of document)

August 2002

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

65

6b. NO. OF REFS (total cited in document)

4

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Defence R&D Canada - Ottawa
3701 Carling Avenue
Ottawa ON K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

1BB22

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

CRC-RP-2002-003

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

DRDC Ottawa TR 2002-073

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNLIMITED

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

U) This final report provides a record of the results of the network management portion of the Sub-Network Access Control Technology Demonstrator project. It summarises the concepts behind this development effort, and describes the completed design and implementation work. This project is part of an on-going effort to build an IP based network using wireless technologies. However, the goal behind this project remains the production of an exploratory prototype. In other words, it is one more step towards the goal of transitioning wireless technologies to the Canadian Operational Fleet.

(U) The proposed IP network presents several unique challenges to network management, due to its low bandwidth wireless links and continually changing topology, that existing products have not been designed to handle. These challenges are explored in order to provide a better understanding of the requirements they impose on network management. A network management tool design is then proposed and the implementation of its prototype is described.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Network Management
Simple Network Management Protocol (SNMP)
Wireless links